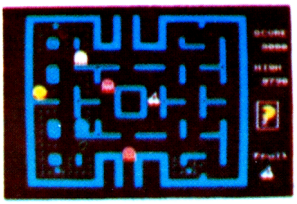
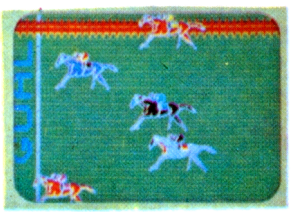
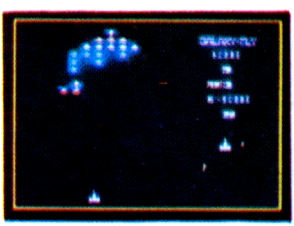
apple II

30.3FFR 30.1200R
30.300R 200.300R
200.2FFR . .
200.3FFR . . .

APPLE II ○







微電腦徹底研究(二)

施 威 銘 著

波前電腦公司出版

APPLE Ⅱ
微電腦徹底研究

(二)

施威銘著

波前電腦公司出版

APPLE Ⅱ
微電腦徹底研究(二)

編著者：施　威　銘

出版者
發行者：波前電腦公司
香港朝光街500號

印刷者：華泰印刷公司
香港仔昌業大廈八樓C座

定價：港幣 $ 25.00

唐凌　序

假如有一天，你用你自己的腦筋，寫了一本書，送到出版社去，或則只是費力地寫好一篇文章，投到某專業性雜誌去發表，你可能遭到一項極大的困難——出版商或雜誌的編輯會向你索取「原文」。

出版商或雜誌編輯之希望文有出處的心理，是不難理解的，一是沾光，一是如有謬誤可一推了之。

事實上，這樣的心態，普通地存在於現居台灣的每一個人心裡，而不只是出版界。

做事有個依據是對的，然而任何依據本身不可能是眞理，即或是眞理，在被引用的過程也可能造成扭曲。

下面就是一段隨處可見而扭曲得可笑的例子：

「在半計數（Counter halves）之間，外來的顫動（Jumper）必須供給。

「若經由Clock 1進入，而且Q 8跳躍至Clock 1，計數比重為1－2－4－5，Q 1是位元比重最大的輸出而且在輸出端有一個對稱的方波。……

這是一段對 7490 動作的描述，如果你對 7490 毫無概念，那麼看了這一段，眞的會令你「顫動」不已。

不幸的事情是，對這樣荒唐的謬誤，出版商給予容忍，讀者也同樣給予容忍。於是成千上萬的學子們，只好一天天地「顫動」下去。

另外一種情況也是很可悲的，也就是不知從什麼時候開始，大學生都喜歡抱厚厚的洋裝書，使得一些類如三用表用法、OTL、OCL 設計或 CMOS 玩具製作之類的小書，有興趣的人只好偷偷地看，總覺得這些小書，好像很不入流一樣。

我與施威銘先生只是初識，當我看完「APPLE Ⅱ微電腦徹底研究」第一册之後，覺得與我主編的「音響技術月刊」頗有其臭相類之處，那就是作者躍然紙上，隨時與你共處。欲這樣，顯然是需要一些勇氣的，一些面對可能的疏失的勇氣，一些面對抱洋裝書的人的勇氣。我未嘗聽說有人照著食譜做菜，把菜燒焦了、燒爛了、糊了，會找作者理論一番的，但如果你寫技術的書、電腦的書，隨時你都得迎接焦了、爛了、糊了的挑戰。

總之，這不是一篇「序」，而是一點感想，我由衷地希望，每一位讀者都能珍惜作者施威銘先生在這本書中「掏」給你的東西。

序　言

這本書是配合APPLE Ⅱ徹底研究㈠而寫的，如果讀過第一册之後再閱讀本書，那麼我希望至少能夠提供您幾點幫助：一對APPLE的瞭解更加深入。二增進編寫及解讀機械語言程式的能力。三彈性運用電腦內藏的系統功能。四徹底明白事物原理後的樂趣。

第0章除了爲引起讀者興趣之外，也指出了COUT、READ等副程式的重要性，如果對GAME等程式保護方法有興趣的話，可以直接閱讀附錄A至F，連同第0章本書總共列了9種磁帶程式保護的方式。第一章是COUT副程式的詳細探討，唯有完全明白COUT的運作原理，我們對APPLE的畫面才能完全掌握。第二章便是模仿第一章APPLE TEXT螢幕的處理技巧來處理高解像畫面，我們發現可以很方便的混合文字、低解像方塊、高解像

圖形於同一畫面上。第三章是說明如何運用& 向量來造出新的BASIC 指令，同時也嘗試造出一個小小型的BASIC 解譯程式，當然這個解譯程式不獨可以由& 來使用，利用CALL 或COUT 讀者也可以自行造出同樣的效果吧！第四章是將READ、WRITE等串行資料處理副程式徹底的加以瞭解，同時也介紹了其他幾種微電腦的讀寫處理方式，除了相互比較之外，也可以做爲程式編寫的參考。

總之，進入第二册之後，我們發現APPLE可以做更多更有用的事，也更徹底的瞭解電腦內部的運作情形了。個人能力有限，內容之表達、校訂多有不完善處，如蒙高明不嫌指正，則必感激不盡。將個人的樂趣分享每一個人，該是本書的目的吧！請大家愉快的來讀這本書！

作者　施威銘

目　　錄

第〇章　30·3FFR

0 — 0　30.3FFR ……1
0 — 1　COUT ……3
0 — 2　READ ……5
0 — 3　保護與解法……7

第1章　COUT徹底研究

1 — 0　COUT　字元輸出副程式……9
1 — 1　COUT　的應用……12
1 — 2　COUT 1 ……16
1 — 3　VIDOUT　的零頁位址 ……19
1 — 4　VIDOUT　的基本副程式單元 ……21
1 — 5　VIDOUT　總整理 ……30
1 — 6　CR　副程式……32

第2章　蘋果與教授

2 — 0　蘋果與教授 ……38
2 — 1　高解像畫面提要 ……39
2 — 2　用 PRINT 指令在高解像畫面印字 ……48

2—3 在高解像畫面造出低解像圖形……59

第3章 創新BASIC指令

3—0 "&"與TXTPTR ……68
3—1 SWAP——APPLE Ⅲ？……69
3—2 GOTO A與GOSUB A ……73
3—3 SOUND、BELL及其他 ……78
3—4 創新BASIC指令 ……80

第4章 WRITE與READ徹底研究

4—0 硬體綫路：$C020與$C060 ……91
4—1 串列訊號的處理……95
4—2 常用的卡帶讀寫方法……97
4—3 APPLE 的 WRITE……103
4—4 HEADR ……107
4—5 APPLE 的READ ……115

附錄A 30.300R （金甲蟲）

A—0 30.300R……123
A—1 程式檢討……124
A—2 保護方式解說……126
A—3 還有一層保護……128

附錄B 30.1200R （大進擊 Ⅱ）

B—0 30.1200R ……131

B—1　程式檢討……132
B—2　保護方式解說……134
B—3　盲目拷貝行不通……135

附錄C　200.300R　（戰略作戰）

C—0　200.300R ……137
C—1　程式檢討……138
C—2　保護方式解說……143

附錄D　200.300R　（攻陷地球）

D—0　200.300R ……145
D—1　保護方式解說……148

附錄E　200.2FFR　（世界末日）

E—0　200.2FFR ……151
E—1　程式檢討……153
E—2　保護方式解說……155

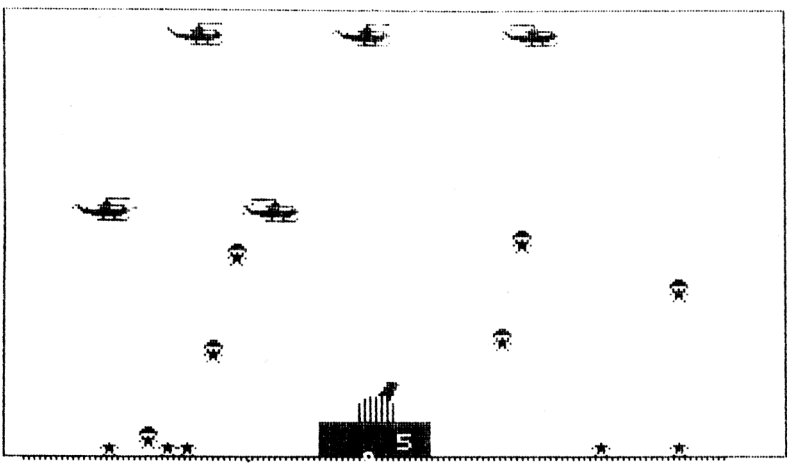
附錄F　3種BASIC保護法

BASIC　Ⅰ　POKE　2049,0 ……157
BASIC　Ⅱ　POKE　214,128 ……158
BASIC　Ⅲ　POKE　82,213 ……158

附錄G　6502　指令表 …… 160

30. 3FFR

—— 傘兵部隊 ——



0—0 30・3FFR

在第一册當中，我們說明了第 2 頁區的功能，因此譬如 200・1200 R這一類的程式保護，相信大家已清楚的瞭解其運作方式而運用自如了。至於其它的保護方法呢？此處讓我們來看 3 0・3 F F R這種程式……

將程式以 3 0・3 F F R讀進 A P P L E裡頭，結果與第2頁區的功能一樣，會自動執行程式，這樣的功能必然是如 200・1200

一般，使用了APPLE的特殊系統結構才能達成，而APPLE的特殊系統使用位址是如附錄所列出的第0頁、第1頁、第2頁、及部份的第3頁位址，而30•3FFR竟然涵蓋了大部份的這些敏感位址，因此解決的第一步是仿照200•1200的方式，先將程式存到自由使用的記憶位址（避開上述的敏感位址），然後將讀進來的資料列出，再慢慢解讀！譬如，我們可以將30•3FFR存到1030•13FFR，如此只要將最高位元的"1"看成"0"，則與原位址的對應也就非常明顯了。現在將市售的一種稱爲"傘兵部隊"的GAME讀到$1030～$13FF位址區間來，要記得在上一册書裡我們已經知道，現在讀進來的是保護程式而不是GAME程式本身。那麼，結果是否也是使用第2頁區的方式來保護呢？將對應於第2頁區（$1200～$12FF）的資料列出來檢視：

```
*1200.129F

1200- FF FF FF FF FF FF FF FF
1208- FF FF FF FF FF FF FF FF
1210- FF FF FF FF FF FF FF FF
1218- FF FF FF FF FF FF FF FF
1220- FF FF FF FF FF FF FF FF
1228- FF FF FF FF FF FF FF FF
1230- FF FF FF FF FF FF FF FF
1238- FF FF FF FF FF FF FF FF
1240- FF FF FF FF FF FF FF FF
1248- FF FF FF FF FF FF FF FF
1250- FF FF FF FF FF FF FF FF
1258- FF FF FF FF FF FF FF FF
1260- FF FF FF FB FF FF FF BF
1268- FF FE E9 F9 BA AB FF FF
1270- FF FF FF FF FF FF FF FF
1278- FF FF FF FF FF FF FF FF
```

結果，我們並未發現存在任何有用的ASCII 指令* ！也就是說第2 頁區內並未存有保護程式。其它位址呢？第3 頁只不過是一些中斷或重置指位器，而第一頁是堆疊暫存區，因此最具可能性的該是0 頁位址區了！可是0 頁是最爲複雜的一個頁區，如何找出問題的所在呢？

0-1 COUT

APPLE的零頁位址有一對指位器（CSWL、H）=（$36，$37）與程式的自動執行有關，這對指位器是由監督程式內的COUT副程式來使用，每次卡帶程式讀入完畢後，便是以JMP COUT來令喇叭發出聲音並且使游標重現。而COUT所去執行的位址，則又由（CSWL、H）來指向，因此，只要GAME程式進入位址存入（$36，$37），那麼一旦程式讀回完畢COUT被呼叫時，便會跳往程式進入點，GAME便自動執行了。我們可以先寫下一小段程式如下：

---

* 見徹底研究㈠第0 章

```
*300:20 DD FB 4C 00 03

*300L

0300-   20 DD FB     JSR    $FBDD
0303-   4C 00 03     JMP    $0300
0306-   FF           ???
```

鍵入 300G 看看，這是連續使喇叭發聲的程式，RESET之後，將 $0300 存入（ $36 ， $37 ），並且連同主程式存入到卡帶上：

```
*36:00 03 N 30.3FFW
```

當程式存入完畢後，程式會自動執行，這是因爲（ $36 ， $37 ）已經指向 $0300 ，當程式存完之後游標重現時，COUT 被呼叫，於是程式便執行了。RESET之後，再將程式讀回：

```
*30.3FFR
```

結果如何？是否與GAME的程式一樣效果呢？如果將主程式裡的喇叭發聲副程式BELL 1 = $FBDD改成 BELL = $FF3A ，結果發現程式不能動作，這是因爲BELL副程式使用到COUT ，而COUT改被改變了方向，功能亦已改變，所以便達不到預期的結果。

如果是欲將存入卡帶的資料先在$1030至13FF的位址區間安排完畢，再存入卡帶，結果應該也可以動作吧？

```
*1036:00 03
*1300:20 DD FB 4C 00 03
*1030.13FFW
*30.3FFR
```

當由卡帶讀回程式時，發現系統似乎失去控制了，爲什麼呢？因爲零頁位址內有許多由系統來使用的指位器、暫存器*、旗號指示器等等，現在除了（$36，$37）是存入指定的數值外，其他位址則是存入$1038至$10FF的一些亂數，當然系統會當掉了。譬如（$3C，$3D）及（$3E，$3F）這兩對指位器……。

0-2 READ

再回到"傘兵部隊"的GAME來，難道這麼複雜的程式竟然可以存放在$30至$3FF不到1K的區間嗎？那麼依照上一節的結論來看看（$36，$37）所指的程式進入點在何處吧！

---

*是指暫時儲存資料用的記憶體，不是CPU內部的暫存器。

```
*1030.10A0

1030- 00 00 FF AA 08 28 00 1D
1038- 81 9E 28 90 3C 00 00 90
1040- 30 00 00 98 2D 98 AE D8
1048- 00 B7 00 00 00 00 3E 45
1050- 69 FF 55 52 00 0D 33 08
1058- 00 00 00 00 00 00 38 DE
1060- 00 00 00 00 00 00 00 01
1068- 08 44 08 44 08 44 08 00
1070- 96 FF 95 00 96 0A FF 00
1078- 00 28 08 00 00 00 08 00
1080- 00 00 00 00 00 00 00 FF
1088- 00 00 00 00 00 00 00 03
1090- 4C 46 00 00 00 00 00 00
1098- 00 00 00 00 00 88 FF FF
10A0- FF
```

現在觀察$1036，$1037這兩個位址，意外的，程式進入點竟然是$1D00！那麼程式是如何存入的呢？

（$3C，$3D）及（$3E，$3F）這兩對指位器是用來做爲卡帶讀入的起始及終止位址指示用的。由上表　可以看出（$3E，$3F）現在所存的是$00與$90，原來程式讀入到（$3E，$3F）時便將$9000存入這對指位器裡，如此一來程式就不止存到$3FF而已，必須存到$9000才算完整。現在問題就很明顯了，程式事實上是存在於$30至$9000之間，而進入點是$1D00。但是存放程式讀入起始位址的（$3C，$3D）呢？爲什麼這個指位器是指向$003C而不是$0030呢？這是因爲如果程式剛好讀入到（$3C，$3D）位址時，而把$0030存入其中的話，這等於是將（$3C，$3D）又指向$0030，如此APPLE會將接下來的資料又重新由$0030開始來存放了，因此必須存入

$003C才可以使接下來的資料依序讀入。

現在已經知道程式的起始、終止位址以及進入點了，那麼如何來解開這種保護呢？

0—3 保護與解法

如果只是爲了拷貝，那麼方法很簡單，避開零頁然後以1030·A000 R來將原來由$30至$9000的程式讀至APPLE裡頭，接着以1030·A000W便可以複製程式了，當然執行時還是要以30·3FFR來讀入才行。但如果是要觀察程式內容的話，那就必須將有關RESET指位器$3F2、$3F3、$3F4等位址加以修改才行，否則於操作當中可能會使程式破壞而須重新再讀入一次。

這一種連主程一齊只有一段的卡帶程式，甚至不必加以解讀其位址與進入點也一樣可以拷貝，只要避開0頁然後以最大的容量830·BFFFR來讀入便可以了，雖然最後可能出現ERR字樣，但是程式仍然是可以正確讀進來的。同理，譬如第一册介紹的200·1200R那種具有兩段錄音的程式，可以用800·1800R 1801·BFFFR來讀入程式，而以800·1800W 1801·BFFFW來複製，當然複製程式仍然是要以200·1200R來執行。於是拿來一捲稱爲"大進擊"的卡帶以1030·BFFFR讀入，然後1030·BFFFW複製了一捲卡帶，最後以30·3FFR執行，結果……？

卡帶的保護方式很多，除非徹底瞭解其原理，否則可說是解不勝解，對上述那種不經解讀程式而直接拷貝的方法，如果程式

是三段的話？四段的話？甚或是以稍異於APPLE的讀寫副程式來存取磁帶資料的話？盲目的拷貝是行不通的，對於這些問題有興趣的讀者，請直接閱讀本書附錄A至附錄F的8種保護法及例子。

1 COUT 徹底研究

在第 0 章的 GAME 我們發現了 COUT 這個很有用的副程式，在本章當中，我們更進一步的來探究其原理及應用，並且看看對於這樣的一個 APPLE 的副程式，我們可以開發應用到何種程度 ……

本章說明字元輸出副程式 COUT 的功能及使用例，由於這個副程式不但是控制了 APPLE 字元輸出的方向，其內容更包含了整個 APPLE 螢幕的印字功能，這些功能包括印字、換行、字幕格式等多項功能，因此值得詳細研究而加以發揮應用。

更重要的是，希望經由瞭解 COUT 運作情形後，能夠利用來做爲我們下一章要談到的高解像畫面處理的工具，對於 APPLE 的充份應用，可說是絕對必要的。

1—0 COUT 字元輸出副程式

APPLE的監督程式是其核心所在，自然是內容豐富，詳加研究可獲益良多。譬如進入位址爲$FDED的COUT這個副程式……

COUT：字元輸出副程式

當APPLE要將一個字元送往週邊設備時，便要使用COUT這個副程式來完成字元輸送的工作。這裡所指的週邊設備是指螢幕記憶位址、週邊插座位址及喇叭發聲副程式位址（BELL1 = EBD9，BELL2 = $FBE4），其動作過程簡繪如下圖：

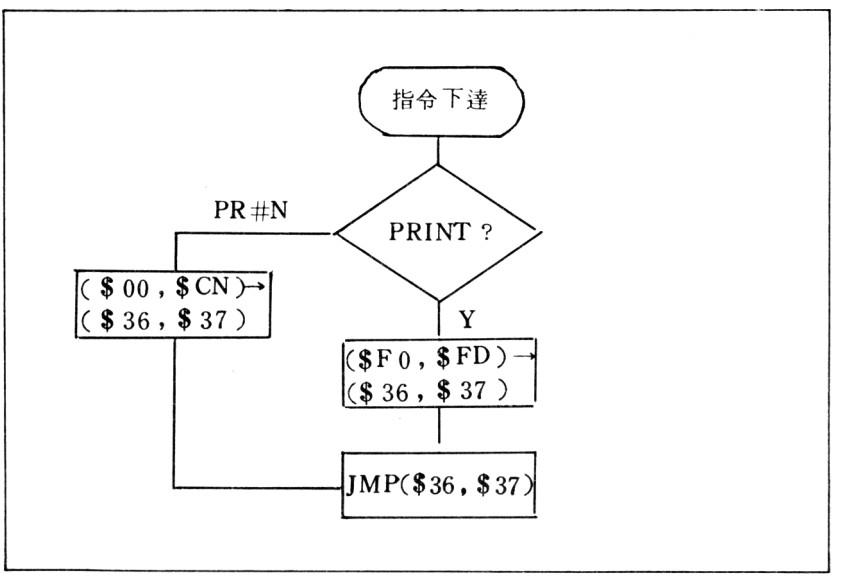


圖 1 — 0

因此，無論是PRINT " ABC "、PR#1、PRINT　CHR$(7)、程式執行完畢後游標要出現、錄音完畢喇叭要發聲甚至是RETURN均要經過COUT副程式。因此，我們便可以瞭解到爲何有些GAME卡帶　當程式讀入完畢之後，喇叭也不發聲便自動執行了。這是因爲一旦程式讀入完畢，便要經由COUT到BELL1副程式，但COUT已被改變方向而直接跳至程式進入點來執行了。

CSWL、H：字元輸出指位器

但是，COUT是如何來往各不同的週邊設備輸送字元呢？COUT的第一句話是：

```
FDED: 6C 36 00          JMP  (CSWL,H)
```

由上式知道，只要程式執行到$FDED（呼叫COUT），便以間接位址法跳往（CSWL、H）所指的位址上去執行程式，因此只要改變CSWL、H的內含值，COUT執行的路綫便不相同了。（CSWL、H）是零頁的（$36，$37）兩個位址，通常APPLE是以螢幕做爲字元輸出的設備，要將字元送往螢幕的副程式是稱爲COUT1的副程式，因此在正常的情況下，（$36，$37）這對指位器的內含值是$F0與$FD，也就是COUT1的進入位址。

COUT1是用來將字元送往螢幕或喇叭的副程式。所以平常使用PRINT等指令時，便會呼叫到COUT，然後經由（CSWL、H）跳往COUT1將要印出的字元碼送往螢幕記憶位址（$400至$7FF），這時硬體綫路便來讀取此記憶位址的字元碼，並且

將之以字形的方式來表現到螢幕上去，這便是APPLE文字幕 的運作過程。

1-1 COUT 的應用

那麼，試試改變COUT的功能，要設定COUT的功能，當然是要由改變（ CSWL、H ）指位器的內含值來達成，首先由BASIC來着手，假如系統的第 1 號週邊插座（要記得週邊插座是由第 0 號算起的 ）已裝設有周邊設備的話，請鍵入：

```
]POKE 54,0 : POKE 55,193
```

結果在第 1 號的週邊設備便會被觸動，在這個例子裡是使用了EPSON的印字機，因此印字機的印字頭便移 動 了 。 這 個過 程的細節是這樣的：當由鍵盤送入上述指令並且按了RETURN 之後，（ CSWL、H ）便被存入 $00 與 $C1 而指向 $C100，這正好是第 1 號週邊插座的位址，因此當執行完 POKE 指令之後，本來要經由 COUT 送往螢幕下兩行最左端位置的游標方塊，結果被送往由（ CSWL、H ）所指向的 $C100 去了，因此印字機便被觸動了。

上述的這個動作與BASIC的PR＃N指令有相同的效果，原來當 PR＃N 指令下達之後，（ CSWL、H ）就被指向 $CN00 的位址，也就是將所對應的週邊位址 $CN00 存入到（$36 ，$37）裡頭。現在鍵入：（ 如果第一號插座接有週邊設備的話 ）

```
]PR#1:PRINT:PRINT PEEK(54),PEEK(55)
2                193
```

依我們所瞭解的，螢幕或印字機應該印出 0 與193 ，爲什現在是印出 2 與 193 呢？這是因爲呼叫了週邊裝置之後，已經過一連串的初值化動作，週邊進入點已經改成 $C102 了（此處是用EPSON印字機界面，若是其它機器，則又不相同）。

有一個例外是 PR ## 0 ，因其結果是指向螢幕而不是週邊插座，因此（ CSWL、H ）指向 COUT1 的位址 $FDF0 ，而不是 $C000 。

但是如果將 CSWL、H 指向其他位址呢？請試着鍵入：

```
]POKE 54,0:POKE 55,3
```

結果系統好像當掉了，就像對一個空的週邊插座下達 PR # N 的指令一般，RESET 以後系統功能又恢復了。前面提到凡是字元輸出就要用到 COUT 就是游標"コ■"也一樣，所以將 $300 POKE 到（ CSWL、H ）按下 RETURN 之後，COUT 便跳到由（ CSWL、H ）指向的 $0300 ，所以結果不是到 COUT1 來印字而是跳到 $0300 去了，這時如果 $0300 沒有存入任何有意義的機械語言程式，那麼系統便停掉回不來，同時游標也不能出現在螢幕上了。同理這便是爲什麼如果第 N 號週邊插座是空着的話，PR # N 便會使系統停止工作的原因了，因爲 COUT 跳到 $CN00 ，而 $CN00 根本沒有任何內容存在！

剛才說 $0300 沒有存入任何有意義的程式，所以系統停止工作。現在請鍵入：

```
*300L

0300-   AD 50 C0    LDA   $C050
0303-   20 32 F8    JSR   $F832
0306-   AD 51 C0    LDA   $C051
0309-   20 DD FB    JSR   $FBDD
030C-   20 DD FB    JSR   $FBDD
030F-   4C 00 03    JMP   $0300
0312-   FF          ???
0313-   FF          ???


]POKE 54,0:POKE 55,3
```

結果畫面上出現充滿了@的文字幕在閃動，並且喇叭發生"嘟、嘟"的聲音，這是因爲 $300 至 $311 已經存入一段程式了，這個程式將畫面設爲低解像畫面，並且清除成黑色，接着觸動 $C051 位址成爲文字幕，再 JSR $FBDD 令喇叭發聲，然後再回到 $0300 重覆執行。但是我們並未下達 300G 的指令，這個程式之所以會執行是因爲我們將 $0300 POKE 進入 CSWL、H＝(54，55)指位器內，當按下 RETURN 後，游標本來要經由 COUT出現在螢幕下兩行的左方，但是由於我們已經使用 POKE 指令來令 COUT 的執行方向改變了，於是 COUT 便跳往 $0300 來執行我們的程式。凡此，只要巧妙安排，便可利用 COUT 來造出許多有趣、有用的效果來。現在我們鍵入：

```
*300L

0300-   20 58 FC    JSR   $FC58
0303-   20 0C FD    JSR   $FD0C
0306-   20 F0 FD    JSR   $FDF0
0309-   4C 03 03    JMP   $0303


]POKE 54,0 : POKE 55,3
```

結果，APPLE變成一部TV打字機了。上面程式中，$FC58 是HOME副程式的進入點，$FD0C是讀取鍵盤副程式，而FDF0就是前面提到的螢幕輸出副程式COUT1，請注意，由於COUT的指向已被改變成$0300，所以要在螢幕輸出字元必須直接呼叫COUT1而不能使用COUT。這段程式的作用是由鍵盤讀取按鍵資料，然後直接印出到螢幕上，如此而已。

有一點要注意的是，每次經由RESET之後，（$36，$37）是會被自動的指向$FDF0，因此如果要保持（CSWL、H）指向的程式能夠繼續執行，可以利用到徹底研究第一册4—3節所提到的方法來解決：

```
*300L

0300-   A9 10       LDA   #$10
0302-   8D F2 03    STA   $03F2
0305-   A9 03       LDA   #$03
0307-   8D F3 03    STA   $03F3
030A-   20 6F FB    JSR   $FB6F
030D-   20 58 FC    JSR   $FC58
0310-   20 0C FD    JSR   $FD0C
0313-   20 F0 FD    JSR   $FDF0
0316-   4C 10 03    JMP   $0310
0319-   FF          ???
```

如此便能保證程式不被RESET所阻斷，對於GAME或其它各種程式也可以依此法來達成。

但是有時候只是要借助COUT跳躍（ JUMP ）的功能而已，也就是執行所欲執行的程式之後，字元的輸出要再回到螢幕上來，那就不能只將（ $36 ，$37 ）POKE就了事的，請試着鍵入：

```
POKE 54,105 : POKE 55,255
```

本來經由COUT可以呼叫監督程式（ $FF69 = − 151 ），但是叫出監督程式後却因COUT指向改變而使得螢幕停止工作了，所以必須在呼叫 $FF69之前先將（ CSWL、H ）歸位才行：

```
*300L

0300-   A9 F0      LDA   #$F0
0302-   85 36      STA   $36
0304-   A9 FD      LDA   #$FD
0306-   85 37      STA   $37
0308-   4C 69 FF   JMP   $FF69
030B-   FF         ???

JPOKE 54,0 : POKE 55,3
```

1−2 COUT1

在正常情況下，（ CSWL、H ）是指向 $FDF0的COUT1副

程式，但是 COUT1 是什麼呢？

COUT1：

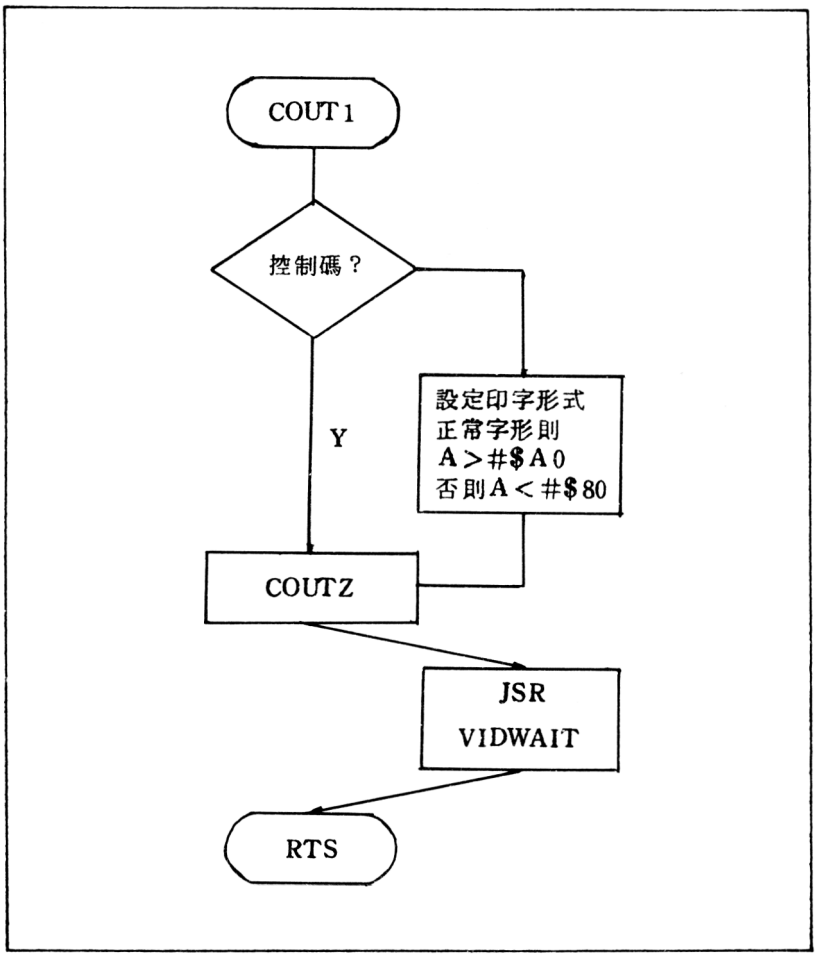


圖 1 — 1

```
(CSWL,H)=($36,$37)
  INVFLG=$32
   YSAV1=$35

FDED: 6C 36 00 COUT   JMP (CSWL,H)
FDF0: C9 A0    COUT1  CMP #$A0
FDF2: 90 02           BCC COUTZ
FDF4: 25 32           AND INVFLG
FDF6: 84 35    COUTZ  STY YSAV1
FDF8: 48              PHA
FDF9: 20 78 FB        JSR VIDWAIT
FDFC: 68              PLA
FDFD: A4 35           LDY YSAV1
FDFF: 60              RTS
```

由上列程式內容看來，COUT1 還不是眞正的螢幕印字程式，眞正的螢幕處理程式是藏在 $FB78 的 VIDWAIT 副程式吧？

VIDWAIT：

```
FB78: C9 8D   VIDWAIT CMP #$8D
FB7A: D0 18           BNE NOWAIT
FB7C: AC 00 C0        LDY KBD
FB7F: 10 13           BPL NOWAIT
FB81: C0 93           CPY #$93
FB83: D0 0F           BNE NOWAIT
FB85: 2C 10 C0        BIT KBSTRB
FB88: AC 00 C0 KBWAIT LDY KBD
FB8B: 10 FB           BPL KBWAIT
FB8D: C0 83           CPY #83
FB8F: F0 03           BEQ NOWAIT
FB91: 2C 10 C0        BIT KBSTRB
FB94: 4C FD FB NOWAIT JMP VIDOUT
```

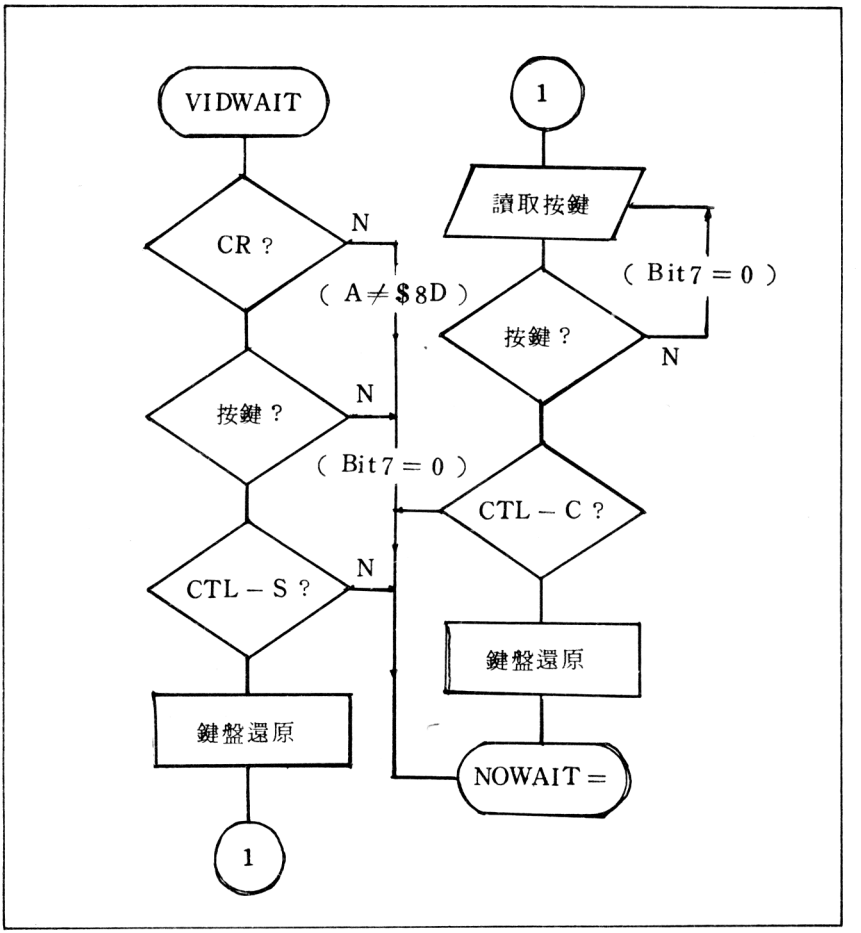


圖1－2　VIDWAIT　副程式

結果VIDWAIT只是判斷輸入的字元碼是否CR、CTL-S或CTL-C而已，我們必須進一步的到VIDOUT副程式去探究其內容，但是VID OUT是比較複雜的牽涉到許多零頁位址及副程式，必須花些時間來詳加研究才行。

1－3 VIDOUT 的零頁位址

零 頁 位 址	
WNDLFT	$ 20
WNDWDTH	$ 21
WNDTOP	$ 22
CH	$ 24
CV	$ 25
BASL	$ 28
BASH	$ 29

表 1 — 0

本節列出與 VIDOUT 相關的零頁位址，並予以扼要說明，記住這些位址的用途，對於 COUT 的解析才能順利進行。

$ 20 = WNDLFT　存放文字窗左端在螢幕上的座標值，其內含值 $ 0 ～ $ 27 。

$ 21 = WNDWDTH　存放文字幕寬度的值，其內含值由 $ 1 ～ $ 28 ，而且與 WNDLFT 之和不可超過 $ 28 。

$ 22 = WNDTOP　存放文字幕頂端在螢幕上的座標值，其內含值由 $ 0 ～ $ 16 。

$ 24 = CH
$ 25 = CV　存放字元與 WNDLFT 的水平距離，COUT 1 要印字時，就是依 CH 及 WNDLFT 的內含值來決定其螢幕水平位置的，CV 作用與 CH 同，但是其內含值決定的是字元輸出在螢幕上的垂直位置。

（$ 28 = BASL
$ 29 = BASH）　存放目前所印字元所在該行的最左端位址，因此以零頁索引定址法將其內含值加上

CH的內含值就是字元碼所應存放的VIDEO RAM的記憶位址了，因此 BASL、H 可稱爲螢幕基綫（ BASE LINE ）指位器，其作用與第一册 2 — 3 節的GBASL、H相似。

1—4 VIDOUT 的基本副程式單元

副 程 式	
$FBF4	ADVANCE
$FBF0	STORADV
$FBC1	BASCALC
$FBD0	BASCLC2
$FC24	VTABZ
$FC22	VTAB
$FC62	CR

表 1 — 1

APPLE 螢幕的各項功能，對於螢幕上的印字如何安排處理，均靠上表的這些副程式來執行，因此可以說是最基本的執行單元了。對於這些副程式，一定要做徹底的瞭解，一方面是爲了能夠順利的閱讀本書以下的各章，另一方也可以學習到螢幕處理的技巧。爲了先掌握住整體概念，首先看這樣一個例子：

```
*300L

0300-   A9 00       LDA   #$00
0302-   85 24       STA   $24
0304-   85 25       STA   $25
0306-   20 F0 FD    JSR   $FDF0
0309-   E6 24       INC   $24
030B-   4C 06 03    JMP   $0306
030E-   FF          ???
030F-   FF          ???
```

這個例子是首先將螢幕起始位置之座標值（0，0）存入（CH，CV），然後到COUT 1（$FDF0）來印出字形，因爲COUT1 每次印字會自動將CH的內含值加一，而我們又在$309 行中將CH的內含值再加一，所以300G之後看到的是間格一行的反白"@"（第一册表 2 — 0 ）。

在這個例子裡要强調的是①COUT1 所印出的是存在於A——暫存器的字元，也就是說印字的過程以LDA來存入字元，然後JSR COUT1 卽可。②經由COUT1 運作之後，A ——暫存器之內含值不被改變，所以印出來的全是字元爲 " $00 " 的反白" @ "。③COUT1 包含自動向右移位、換行、螢幕上捲（SCROLL）等動作。

現在可以開始來瞭解表 1 — 1 中的副程式了，首先由最基本的單元開始：

ADVANCE = $FBF4

```
FBF0: A4 24     STORADV LDY CH
FBF2: 91 28             STA (BASL),Y
FBF4: E6 24     ADVANCE INC CH
FBF6: A5 24             LDA CH
FBF8: C5 21             CMP WNDWDTH
FBFA: B0 66             BCS CR
FBFC: 60        RTS3    RTS
```

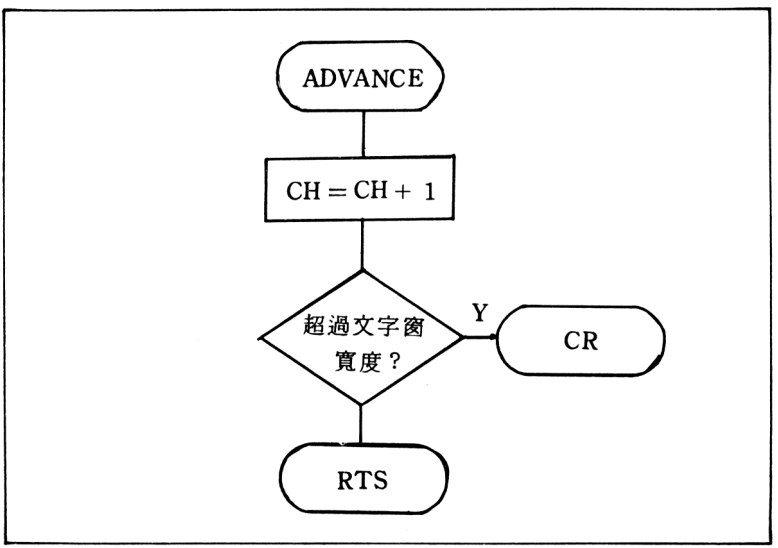


圖　1 — 3

這個副程式是將字元印出水平位置加一，然後檢查是否超過文字窗寬度，若超過則到 CR 副程式，CR 副程式會將字元印出位置移到下一行的最左端，如果已到達最底端，則令螢幕上捲。

STORADV = $FBF0

這是整個COUT程式中，眞正執行印字的副程式，它的工作就是將字元位置的水平距離（CH的內含值）取入Y ——暫存器，然後以基綫指位器（BASL、H）所指的位址爲基準，將暫存器A的值（即所欲印出之字元碼）以零頁索引定址法STA （BASL），Y的方式，存到$0400至$07FF的螢幕顯示記憶區（VIDEO RAM）。這個動作和我們以POKE 1024，193這樣的指令來將字元碼存入螢幕記憶體的效果是一樣的，但是由於有COUT的所有副程式群來支援，所以才有螢幕上多樣的功能。

BASCALC = $FBC1

```
FBC1: 48        BASCALC PHA
FBC2: 4A                LSR A
FBC3: 29 03             AND #$03
FBC5: 09 04             ORA #$04
FBC7: 85 29             STA BASH
FBC9: 68                PLA
FBCA: 29 18     BASCLC2 AND #$18
FBCC: 90 02             BCC $FBD0
FBCE: 69 7F             ADC #$7F
FBD0: 85 28             STA BASL
FBD2: 0A                ASL A
FBD3: 0A                ASL A
FBD4: 05 28             ORA BASL
FBD6: 85 28             STA BASL
FBD8: 60                RTS
```

BASCALC（Base Calculate）是計算基綫位址的副程

式，其作用是將螢幕垂直座標值換成對應的記憶位址，由於位址有16個位元，因此總共需要（ BASL、H ）這樣一對指位器來存放，而且因爲螢幕範圍是由$0400至$07FF，所以BASL的內含值可以由$00～$FF，但是因爲只要螢幕最左端位址，所以也要經過詳細計算。而BASH則一定要在$04至$07之間，若大於$07則會破壞BASIC程式，小於$04則會影響系統正常運作。通常是以LDA CV的方式將垂直座標值取入A暫存器，然後再送入BASCALC計算，因此進入BASCALC時，A的值是介於$0～$17之間（0～23之間）。現在接着來看看如何計算出（ BASL、H ）要明白這個問題，必須要回到第一册2—2節，文字幕構造的公式（ 2—1 ），現在重寫如下：

```
B = 1024 + X + ( A/8 ) * 40 + ( A MOD 8 ) * 128 .....
.....式( 1—0 )
```

上式是以INTEGER BASIC寫成，A代表螢幕上第A列，也就是暫存器A的內含值，B是眞正的記憶位址，X代表該列上第X格。現在以二進位方式來考慮，由於只算Y座標值，因此第二項的X必須由（式1—0）中去除。第三項的（A/8）* 40是以LSR將A右移三次，然後再以ASL左移三次，最後再乘5（即ASL兩次再OR本身）。第四項是A除以8之後的餘數乘以128，也就是A AND ＃$07之後再ASL 7次。

BASCALC的前半部份是計算BASH，因此是螢幕位址

的高 8 位元，能夠進入高 8 位元的項只有第一項1024 = $0400 及第四項，而第四項進入高 8 位元的是 A 暫存器的第 1 及 2 兩位元，現在圖示如下：

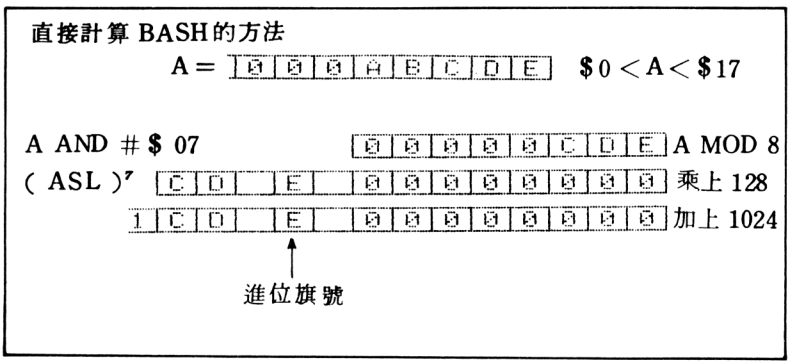


圖 1—4

因此最後的結果只要造出一個位元組其第0、1兩位元是A暫存器的第 1 、2 兩位元，而第 2 位元爲 1 ，其它爲 0 便可以了。 所以將 $FBC1 的 BASCALC 分解開來，便可看出結果與前面的直接計算結果一樣，當然步驟是簡化了。

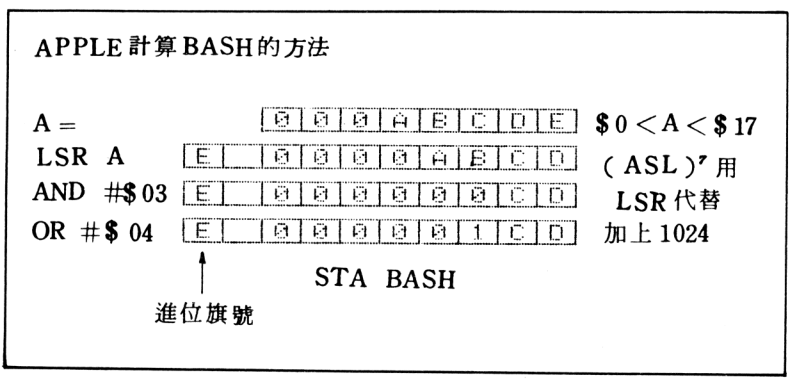


圖 1—5

BASCLC2 = $FBCA

在BASCALC設定了BASH，接下來就由後半段的BASCLC2設定BASL，其原理已如上述。但是，原資料（APPLE REFERENCE MANUAL）對於BASCLC2的標名（LABEL）放置的位址似乎不妥，應該是放在$FBCA才較妥當（原資料置於$FBD0）。那麼，BASL又如何計算的呢？參看圖1—4可以知道BASH只取高八位元，那麼BASL就是低八位元了，對低八位元有影響的只有（式1—0）的第三、四兩項，現在圖示如下：

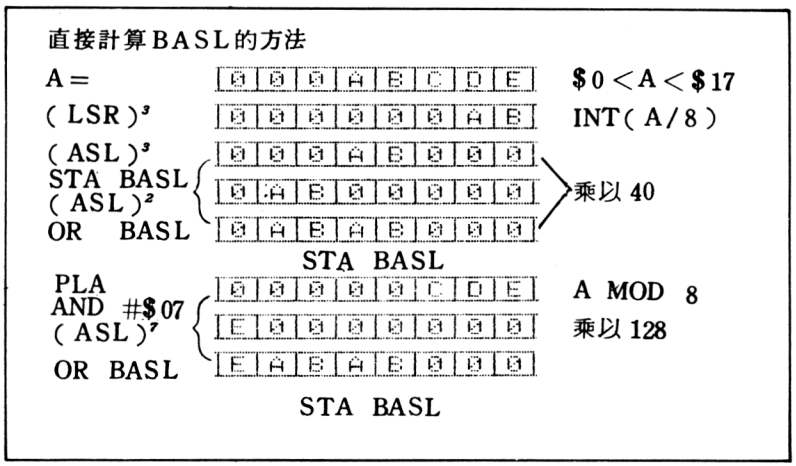


圖 1—6

現在來看上述的直接計算APPLE是如何簡化的：

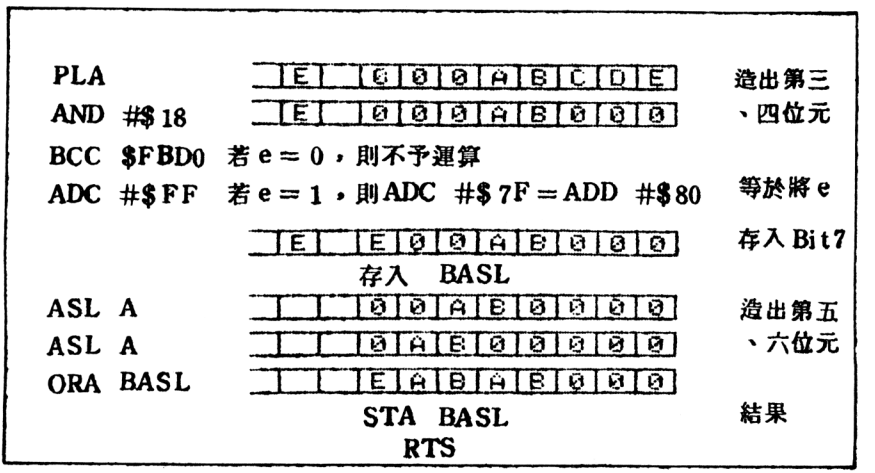


圖 1－7

所以，（BASL、H）的值就由BASCALC來計算完畢。接下來，就看這個副程式如何使用……

VTABZ＝$FC24

```
FC22: A5 25     VTAB    LDA CV
FC24: 20 C1 FB  VTABZ   JSR BASCALC
FC27: 65 20             ADC WNDLFT
FC29: 85 28             STA BASL
FC2B: 60        RTS4    RTS
```

螢幕水平位置很容易計算，只要將（ BASL、H ）的值算出，加上Y暫存器的值就可以了，其實這個動作就是由STORADV來完成的，在第一册 2 — 3 節的低解像畫面我們也看到類似的情形。但是垂直位置的計算就不甚簡單了，如果要畫一條垂直綫的話，則勢必要每畫一點皆要經由BASC - ALC來計算一次，這也便是爲什麼APPLE畫垂直綫比水平綫慢的原因了。

VTABZ這個副程式就是用來處理螢幕垂直位置的副程式，它計算BASL、H之值，但是基準點是以文字窗爲準而不是以螢幕爲準，這是與BASCALC不同的地方。

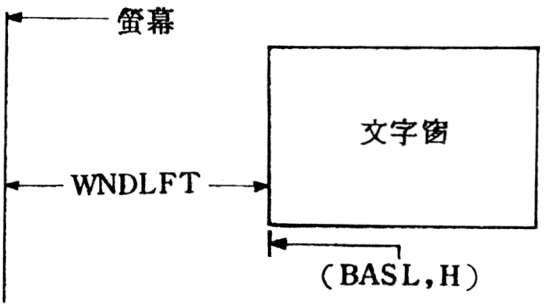


圖 1 — 8

VTAB = $FC22

VTAB 是將 CV 的值取入 A 暫存器，然後經由 VTABZ 來設定（ BASL,H ）的值。可是，如此的話，又何必要標示 VTABZ，只有 VTAB 便可以了吧？這是因爲有時 APPLE 其它副程式或使用者希望能另行設定垂直座標值，而不要經由 CV 設定，這時便可利用 VTABZ 而不必經由 VTAB 了。

1—5 VIDOUT 總整理

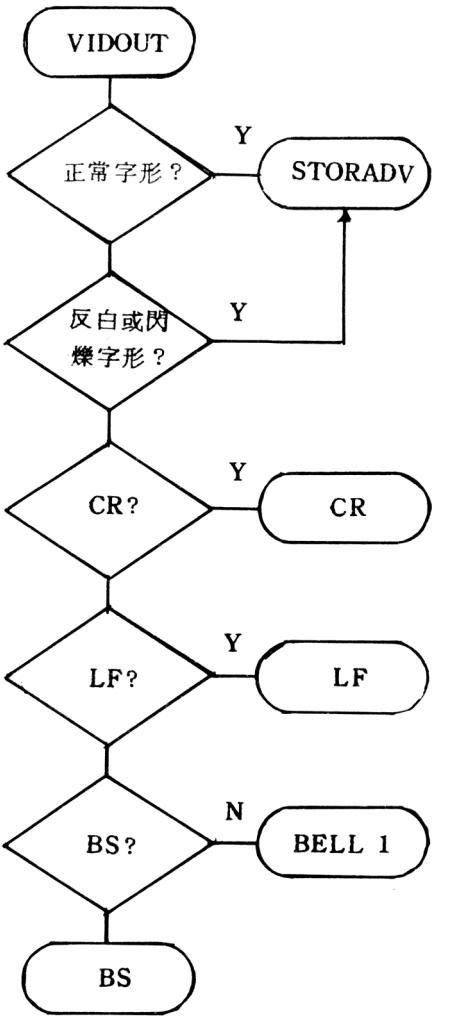
```
FBFD: C9 A0     VIDOUT CMP #$A0
FBFF: B0 EF            BCS STOADV
FC01: A8               TAY
FC02: 10 EC            BPL STOADV
FC04: C9 8D            CMP #$8D
FC06: F0 5A            BEQ CR
FC08: C9 8A            CMP #$8A
FC0A: F0 5A            BEQ LF
FC0C: C9 88            CMP #$88
FC0E: D0 C9            BNE BELL1
FC10: C6 24     BS     DEC CH
FC12: 10 E8            BPL RTS3
FC14: A5 21            LDA WNDWDTH
FC16: 85 24            STA CH
FC18: C6 24            DEC CH
FC1A: A5 22     UP     LDA WNDTOP
FC1C: C5 25            CMP CV
FC1E: B0 0B            BCS RTS4
FC20: C6 25            DEC CV
FC22: A5 25     VTAB   LDA CV
FC24: 20 C1 FB VTABZ   JSR BASCALC
FC27: 65 20            ADC WNDLFT
FC29: 85 28            STA BASL
FC2B: 60               RTS
```

VIDOUT 由 $FBFD 開始，首先以 CMP # $A0 來 判斷是否爲正常字形碼或控制字元（前者大於 $A0 ），若是字形碼，則到 STOADV 將字形輸出，並且將 CH 之值加一使下一個字印出的位置向右前進一格，同時檢查是否該列已填滿至文字窗寬度了，若否則結束，若已填滿則到 CR 做換行或字幕上捲的處理， CR是至目前我們尚未說明的副程式，將要待下一節處理。其中，如果鍵盤送過來的是後退字元（ CTL- H ），則只是將 CH 減一就可以，但是若已退到該列最左端了，那就要由上一列的最右端出現了，而如果已達到文字窗頂端，當然不可再上移了。上移的動作是以 UP 副程式來達成，首先判斷是否已達頂端，若否，則 CV 減 一使印字位置上移一列，然後到 VTAB 完成位置設定，因爲 CH 這時是存入 WNDWTH 減一的值，因此出現在文字窗最右端。

如果鍵盤送來的是小於 $A0 的字元碼呢？由於 COUT1 曾經 AND INVFLG 所以也可能是反白或閃爍字形，而不一定是控制字元因此首先以 BPL 來判斷是否字形碼 *（字元碼 Bit 7 = 0 ），若是（ Bit 7 = 0 ）則至 STOADV 副程式印出，若否，則判斷是否爲 CR 、換行（ LF ）或後退（ BS ），若是則分別到各該副程式處理，若否，則到 BELL1 副程式，請參照圖 1 — 9 ， 1 — 10。

---

* 見第一冊表 2 — 0



1—9圖

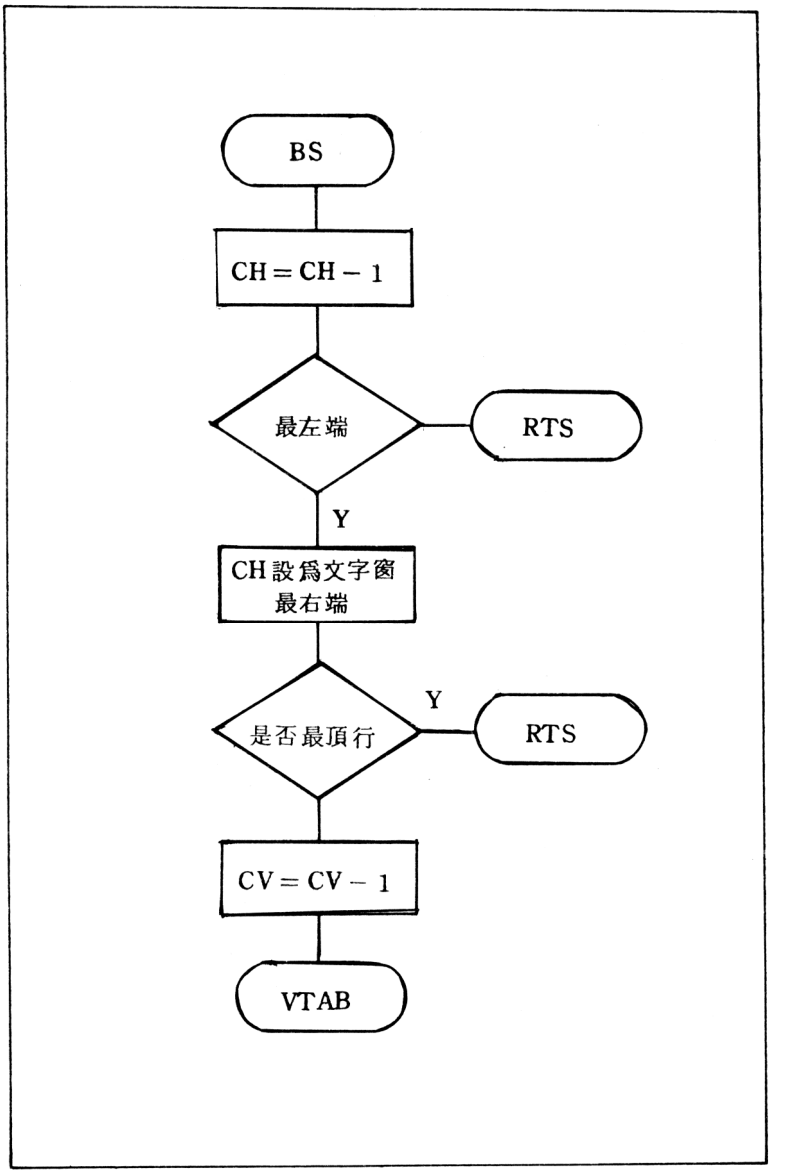


圖　1 — 10

1-6 CR 副程式

與COUT有關的副程式，大致已介紹完畢，最後只剩下CR這個副程式。CR副程式的功用是什麼呢？我們曾經在兩個地方看到CR這個副程式，那就是STORADV副程式向右填字超過文字窗寬度時，或是以RETURN鍵呼叫VIDOUT時，便需要CR來將游標位置移到下一行或下兩行之最左端。

CR 副程式

```
FC62: A9 00       163 CR       LDA #$00
FC64: 85 24       164          STA CH
FC66: E6 25       165 LF       INC CV
FC68: A5 25       166          LDA CV
FC6A: C5 23       167          CMP WNDBTM
FC6C: 90 B6       168          BCC VTABZ
FC6E: C6 25       169          DEC CV
FC70: A5 22       170 SCROLL   LDA WNDTOP
FC72: 48          171          PHA
FC73: 20 24 FC    172          JSR VTABZ
FC76: A5 28       173 SCRL1    LDA BASL
FC78: 85 2A       174          STA BAS2L
FC7A: A5 29       175          LDA BASH
FC7C: 85 2B       176          STA BAS2H
FC7E: A4 21       177          LDY WNDWDTH
FC80: 88          178          DEY
FC81: 68          179          PLA
FC82: 69 01       180          ADC #$01
FC84: C5 23       181          CMP WNDBTM
FC86: B0 0D       182          BCS SCRL3
FC88: 48          183          PHA
FC89: 20 24 FC    184          JSR VTABZ
FC8C: B1 28       185 SCRL2    LDA (BASL),Y
FC8E: 91 2A       186          STA (BAS2L),Y
FC90: 88          187          DEY
FC91: 10 F9       188          BPL SCRL2
FC93: 30 E1       189          BMI SCRL1
FC95: A0 00       190 SCRL3    LDY #$00
FC97: 20 9E FC    191          JSR CLEOLZ
FC9A: B0 B6       192          BCS VTAB
FC9C: A4 24       193 CLREOL   LDY CH
FC9E: A9 A0       194 CLEOLZ   LDA #$A0
FCA0: 91 28       195 CLEOL2   STA (BASL),Y
FCA2: C8          196          INY
FCA3: C4 21       197          CPY WNDWDTH
FCA5: 90 F9       198          BCC CLEOL2
FCA7: 60          199          RTS
```

如果只是將游標移下移至最左端，那麼CR的動作很簡單，

只要將 $00 存入 CH，將 CV 加一，回到 COUT 時便可將游標移至下行的最左端。如果目前游標所在已經是最底下一行，則必須令目前螢幕所有文字上移一行，然後游標移至最左端。程式中（BAS2L、H）是用來做（BASL、H）上 一行的基綫位址指位器。

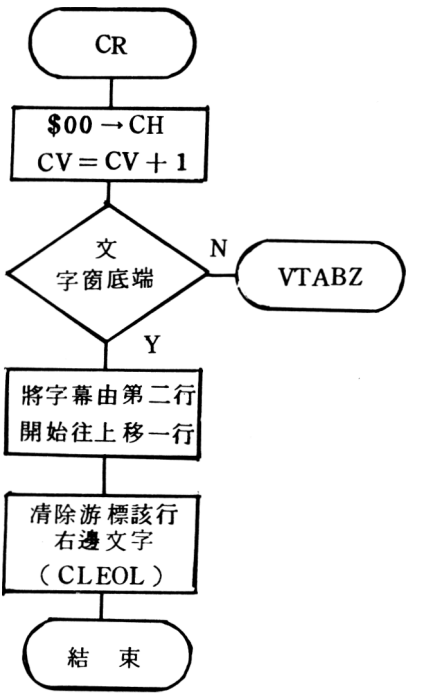


圖 1 — 11A

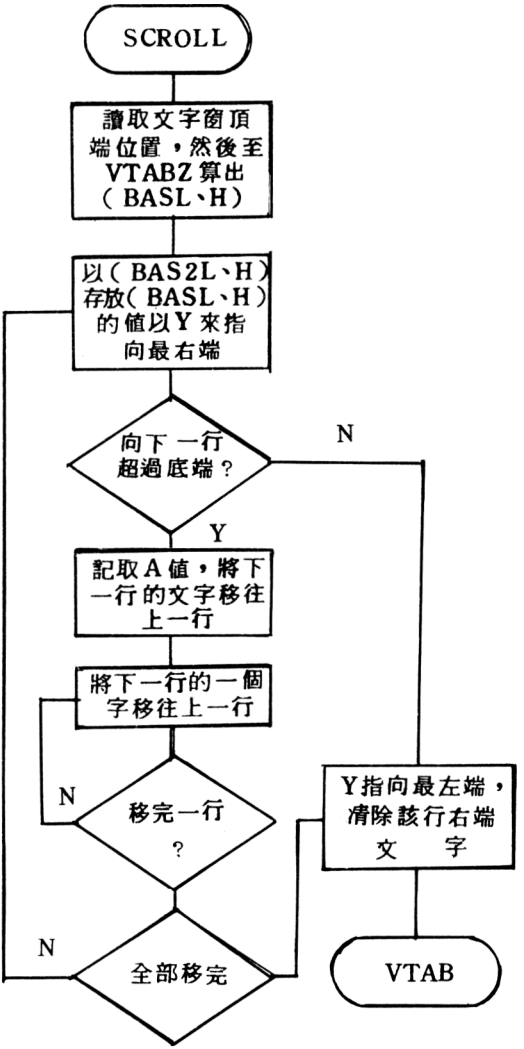
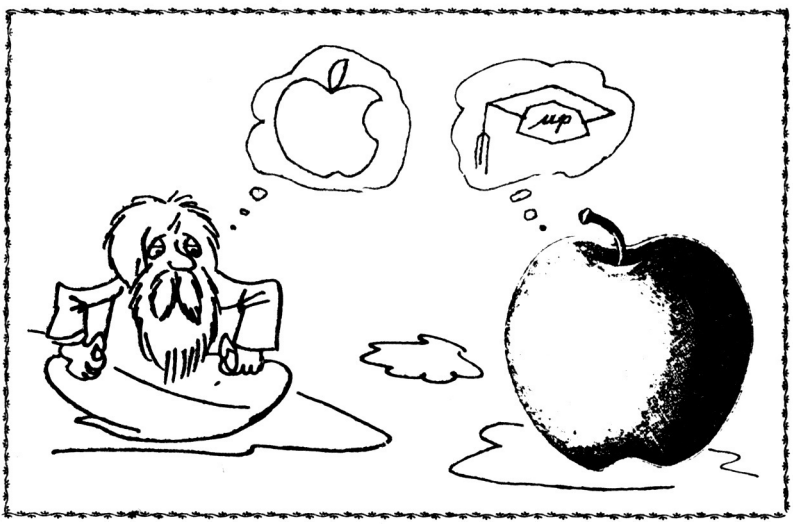

SCROLL
讀取文字窗頂端位置，然後至VTABZ算出（BASL、H）
以（BAS2L、H）存放（BASL、H）的值以Y來指向最右端
向下一行超過底端？
N
Y
記取A值，將下一行的文字移往上一行
將下一行的一個字移往上一行
N
移完一行？
Y指向最左端，清除該行右端文字
N
全部移完
VTAB

圖　1 — 11B

蘋果與教授 2

—— 電腦模擬電腦 ——



目前個人用電腦的機種很多，也各有其優點，什麼樣的電腦才適合使用呢？那可眞是因人、因事而異，沒有一定的答案。常常有這樣的事情發生，在長期的考慮之後，下決定買了一部認爲合適的微電腦，可是過了一段時間，却漸漸發現因某些功能的缺失而使用起來不方便，甚或無法表現出期望的效果來。對於這樣的問題要如何來解決呢？通常這是因爲機器本身設計的限制，如果是對於硬體綫路有適當瞭解的人，那麼他可能動手加以改良，可

是對於缺乏電子知識與經驗的人呢？本章就以一些實例說明如何以軟體上的技巧來彌補並改良硬體設計之不足，同時看看我們在第 1 章提到的：能夠將ＡＰＰＬＥ隱藏的功能開發應用到何種程度………

2—0　蘋果與教授

ＡＰＰＬＥ　Ⅱ這部微電腦的優點之一是它的高解像畫面，在做函數、曲綫之細緻描繪時更具見其特長，但是令使用者常常感到不方便的是在做圖形與文數字混合使用時，便沒有如 PC—8001，VIC—20，ATARI，小教授二號……等的方便了。其中最有趣的是幾乎與ＡＰＰＬＥ合用（Compactable）的小教授微電腦，這部電腦具有幾乎全部的ＡＰＰＬＥＳＯＦＴ ＢＡＳＩＣ指令，而且更具有ＡＰＰＬＥ沒有的“圖形單元”、“圖形與文數字混合”、“單鍵指令”、“中文……”……，竟然這麼多的功能是密集在不到APPLE ⅛的體積內，是如何做得到的呢？

原來小教授的設計者是將ＡＰＰＬＥ的鍵盤、週邊插座、字形產生器、文字幕……等相關的硬體綫路全部去除，然後以軟體程式來達成這些硬體的功能。其中文字幕是以高解像畫面來模擬的，也就是說字形的資料不是存放到＄400～＄7FF的ＶＩＤＥＯ RAM而是存到＄2000至＄3FFF的高解像記憶區內，我們就這個問題在本章討論並試着造出類似或更強化的功能，並且希望達成：①鍵盤按字直接出現在高解像畫面上而同時ＡＰＰＬＥ也能接收這些指令②能以ＰＲＩＮＴ指令印字在高解像畫面上③以COLOR及 PLOT 畫出低解像圖形及“卌 士 目 月”之類的圖形單元

2—1 高解像畫面提要

這是我們一序列高解像公用程式的開始，請每一步驟皆詳予瞭解，……

有關高解像畫面的原理及結構在第一册第 5 章已予詳述，現再將某些必要的部份說明如下：

首先參照表 2 — 0 ，這個表格基本上與文字幕的格式是一樣的，共有 24× 40 個方格 ，只不過在表 2 — 0 的每一方格裡，不是特定的字形而是由高解像的點來組成的，每一方格包含了 8 條水平掃描綫，水平綫由 7 個點組成，每一水平掃描綫對應到一個記憶位址，而該位址所存的 8 個位元除了最高位元不顯示外，其餘 7 個位元就由低位元開始由左至右的對應到水平綫的 7 個點上，所以方格裡共 8 × 7 點，整個畫面就是（ 8×24 ）×（ 7×40 ）= 192×280 點。

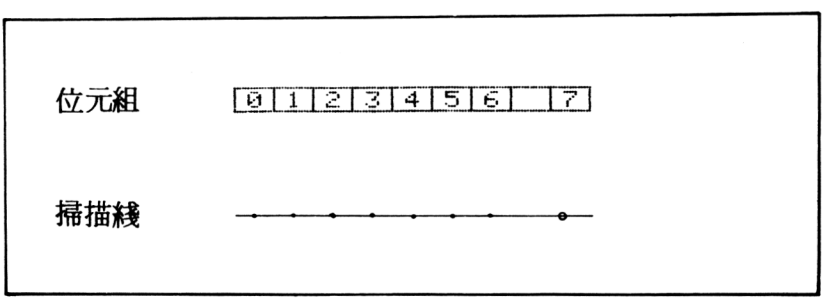
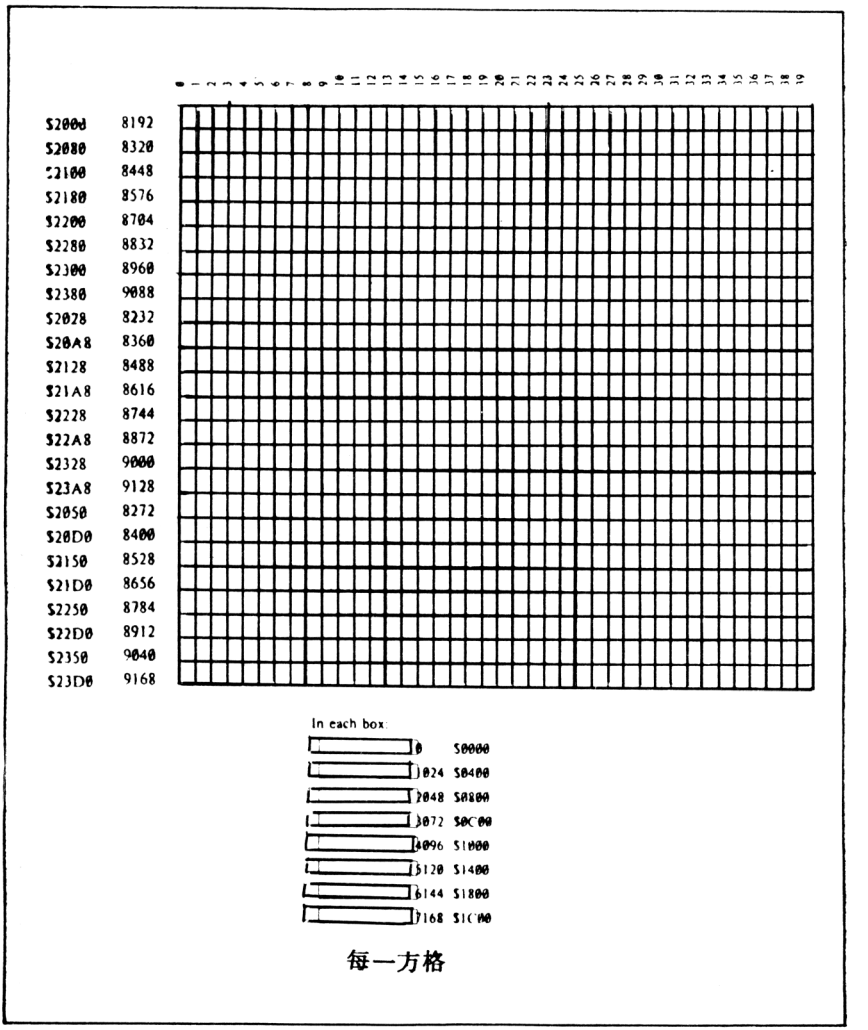


圖 2 — 0



每一方格

表 2－0

接着要知道的是如何將畫面座標換算成記憶位址，這是因爲每次我們要在畫面上某個位置上畫圖形，其實就是將圖形資料存

入該位置對應的記憶位址，所以當然要算出記憶位址與螢幕座標的關係：

$$B=8192+[(A/8)\ MOD\ 8)]*128+(A/64)*40+(A\ \ MOD\ \ 8)*1024+X$$

……（式 2 － 0 ）

上式是以 INTEGER BASIC 來計算的。其中第一項是高解像畫面的起始位址（ $2000 ）的十進位數值 ，第二、三兩項是將表 2 － 0 中每一方格上面第一條掃描綫的螢幕垂直座標值，換算成對應的記憶位址，第 4 項是將方格內掃描綫與第一掃描綫的位址差距算出，最後一項是水平位置，也就是水平的位址距離。因此只要設計出一個程式，將螢幕座標值（ X 、 A ）輸入，然後將對應的位址 B 算出，如此便可以很方便的讓我們在高解像畫面上存取資料了。

但是，高解像畫面不是彩色的嗎？根據 APPLE Ⅱ的手册指出：如果一個點的相對應位元是 " 0 "，則該點不被顯示，如果是 " 1 " 則顯示，但是還要依照下列規則：

(1)在偶數行的點，如果其位址最高位元（即不被顯示的那一位元）是 " 0 " 則爲紫色。若最高位元爲 " 1 " 則爲藍色。

(2)在奇數行的點，如果其位址最高位元是 " 0 " 則爲綠色。若最高位元爲 " 1 " 則爲橙色。

(3)任何相鄰兩點若同時點亮顯示時，則顏色變爲白色。

(4)必須相鄰兩位元同時爲 " 0 " 才會出現黑色。

由上面規則知道是不可能令同一位址上的點同時出現綠色及藍色，綠色及橙色，紫色及藍色或紫色及橙色的。

下圖就是一些例子

0	0	0	0	0	0	0	0	黑色 ＝$ 00
1	0	0	0	0	0	0	0	移相的黑色 ＝$ 80
0	1	1	1	1	1	1	1	白色 ＝$ 7F
1	1	1	1	1	1	1	1	移相的白色 ＝$ FF
0	1	0	1	0	1	0	1	紫紅色 ＝$ 55
0	0	1	0	1	0	1	0	綠色 ＝$ 2A
1	1	0	1	0	1	0	1	藍色 ＝$ D5
1	0	1	0	1	0	1	0	橙色 ＝$ AA
0	1	1	0	0	1	0	1	白色及紫紅 ＝$ 65
1	1	0	1	0	0	1	1	藍色及白色 ＝$D3

圖 2－1

所以只要鍵入：

```
]LIST

10  HGR
20  FOR I = 0 TO 7
30  POKE 8192 + I * 1024,211
40  NEXT
```

程式 2－0

就可以將藍白方塊（ $D3＝211 ）存到螢幕左上角了。請注意，

由於是低位元在左邊，所以藍色在左白色在右。

那麼要如何來運用高解像畫面呢？第1章的APPLE文字幕處理副程式COUT不就是我們最好的例子嗎？這也便是我們要詳細說明COUT的原因，只要依COUT的基本結構，再加以適當的修改，一樣可以用到高解像畫面吧？爲了簡化問題使之更容易明白，我們將COUT的某些部份先予刪除，只取其基本執行單元來使用即可，請參照表1－1COUT的基本副程式單元爲：ST-OADV、BASCALC、VTAB及CR，而相關的零頁位址爲（BASL、H）。

照第一章的說法，STOADV是將字元碼存到$400～$7FF螢幕記憶區的一個位址內，然後由硬體來掃描此一位址並表現成字形，所以只要存一個字元碼即可。在高解像畫面時，如例2—0 POKE方塊的例子，是一次需要8個位元組才能造出一個圖形單元或5×7的點陣字形，這是與STOADV不同的地方，高解像畫面的STOADV副程式我們稱爲HSTADV。

HSTADV

STADV是以經由BASCALC計算出來的（BASL、H）做爲基準位址，然後以零頁索引位址法將資料存入螢幕記憶位址區，但是高解像圖形須要8條綫（位址）才能構成一個字形，那麼HSTADV要如何來處理8個位址的資料呢？程式設計方法很多，一個方法是以高解像基綫計算副程式HBSCALC來算出方塊8條掃描綫的基綫位址（HBASL、H），那麼HSTADV只要依此8個（HBASL、H）的位址來將8個位元組的資料存

入即可，另外一個方法是像第一册例5－3的BASIC程式一般，只要算出第一條掃描綫的位址，然後由HSTADV來將資料依次填入此8個掃描綫位址。但是爲了保持彈性，我們採用另一個方法，那就是HBSCALC每次將一條掃描綫換算成位址，而HSTADV則視需要（由使用者設定）的次數，來呼叫HBSCALC，如果要畫4條掃描綫的低解像方塊，便呼叫4次，如果要畫8條掃描綫的字形，便呼叫8次，依此類推。HSTADV的初步設計如下：

```
HSTADV  LDX #$08
        LDY #$00
HSTAV2  JSR HBSCALC        計算綫的基綫位址
        CLC
        LDA HBASL
        ADC CH
        STA HBASL          水平位址
        LDA HBASH
        ADC #$00
        STA HBASH
        LDA (TABLE),Y      取出第一綫資料
        STA (HBASL,H),Y    存入第一綫位址
        INC HCV            指向下一綫
        INC TABLE          指向下一綫資料
        DEX
        BNE HSTAV2
        RTS
```

程式　2－1

首先是設定畫出的掃描綫數爲 8 條，然後到 HBSCALC（假設已存在）去算出基綫位址，接着將基綫位址加上水平距離，這便是眞正的掃描綫位址了，由字形或圖形 TABLE 取得資料之後就存到該位址上，這時螢幕上就出現對應的圖案，然後計數減一繼續動作至 8 條綫皆完成而結束。由於 LDA、STA 指令只有零頁索引是間接定址方式，因此只好採用間接索引定址法 LDA （TABLE），Y，但是並未用到索引的性質，因此程式一開始 Y 就設爲 0 。

要使用這個程式，必須先將所欲畫方格之位置座標（X，Y）存入 HCV 及 CH，HCV 必須是 8 條掃描綫的第一綫的垂直座標值才行，因此是 0 、8 、16 、24 、……184 ，而 CH 則是由 0 ～ 39 的值，這是因爲雖然 APPLE 有 280×192 的高解像畫，但是其實水平方向只由 40 個位址來對應，每個位址提供 7 個位元的資料來畫點，才構成水平的 280 點（而且只能 140 點同時出現，否則便會連成一白綫無法解析了 * ），至於所要畫出方塊的顏色及形式則由圖形表來設定。

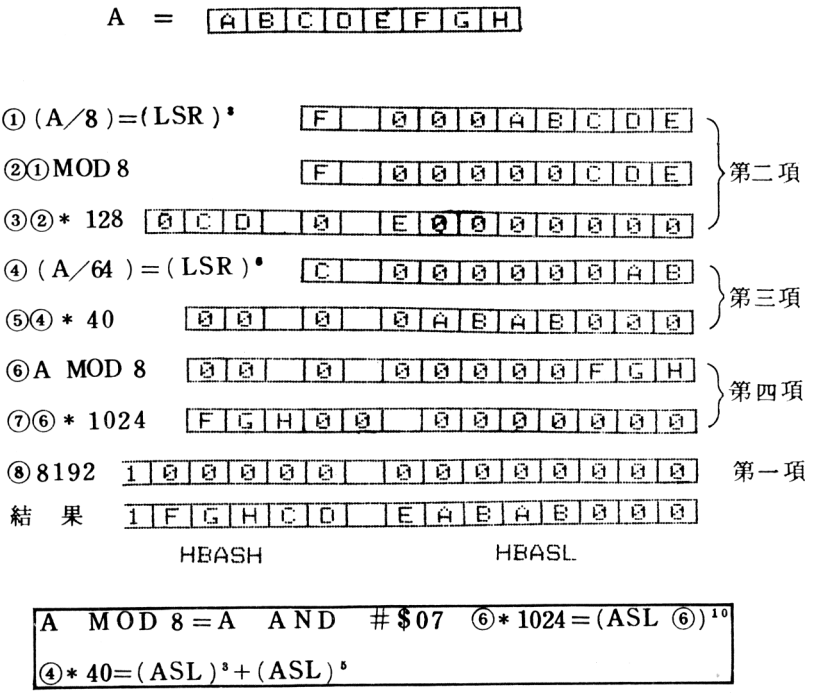
HBSCALC

HSTADV 基本上是完全依靠 HBSCALC 來指定其資料所應存放的位置，因此 HBSCALC 是很重要的部份，而且其內容也較不易理解，我們採用如 1 － 4 節的方式來一步步的說明。

回顧（式 2 － 0 ），位址 B 共由 5 項構成，第 5 項與現在要計算的（HBASL、H）無關，第 1 項 8192 ＝ $2000，也就是在 HBASH 要加上 $20 ，第 2 項是將高解像垂直座標 A 除以 8 後

* 第一冊 5 － 0

再除 8 取餘數，所以是 LSR 三次之後再 AND # $07，顯然的這一項只對 HBASL 有用，第 3 項是 A 除以 64 之後，再乘以 40，也就是 LSR 6 次之後再分別 ASL 3 次及 5 次然後相加（與 1－3 節相似），這一項也只對 HBASL 有用，第 4 項是 A AND # $07 然後 ASL 10 次，因此只對 HBASH 有用。結果整理如下：



A MOD 8 = A AND #$07　⑥* 1024 = (ASL ⑥)¹⁰

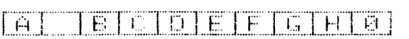
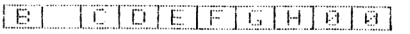
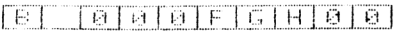
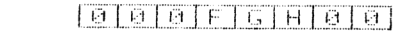
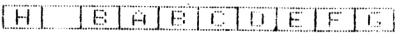
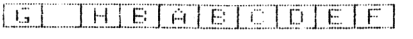
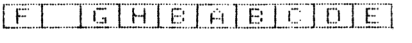
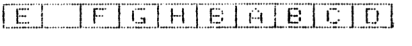
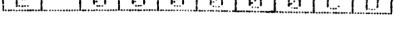
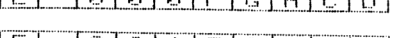
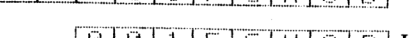
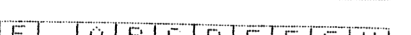
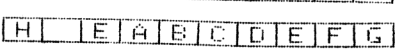
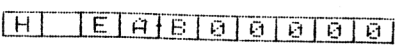
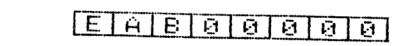
④* 40 = (ASL)³ + (ASL)⁵

圖 2－2

要直接將上述運算以機械語言寫下來佔用很多記憶位址，譬如將⑥乘以 1024 要 ASL 10 次，整個過程將會很長。因此必須如 1

－4節的BASCALC一般的以最少的運算將結果製造出來，只要（HBASL、H）的值與圖2－2一樣便可以了。

首先假設高解像畫面的Y座標值是存在於零頁位址$ 00 ，而以（$14 ，$15 ）當成（HBASL、H），整個程式解說如下：

LDA	$00	
ASL		
ASL		
AND	#$1C	
STA	$15	
LDA	$00	
ROR		
ROR		
ROR		
ROR		
AND	#$03	
ORA	$15	
ORA	#$24	
STA	$15	 HBASH
LDA	$00	
ROR		
AND	#$E0	
STA	$14	

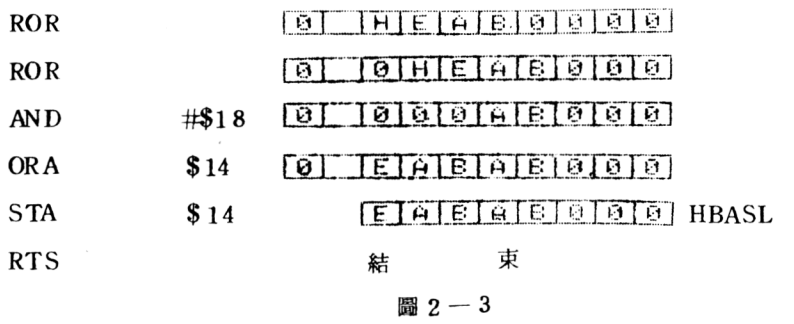


圖 2 — 3

到此，我們已造出將垂直座標換成高解像畫面每列的起始位址的副程式HBSCALC了。只要將垂直座標存入$00經由HSTADV來呼叫HBSCALC便可算出（HBASL、H），然後由HSTADV將圖形或文字資料存入該位址，如此連續8次（可由POKE來改變次數）便可在螢幕上造出各種文字或圖樣了。

目前唯一還沒做的就是查表程式了，我們的目標不但要造出文數字，還要造出圖形單元以及低解像方塊，因此要將字元碼詳細分配才行，首先當然是以最重要的文數字開始……

2—2 用PRINT 在高解像畫面印字

我們的一個目標是，能夠將APPLE的高解像畫面改良成如同小教授2號電腦一樣來使用，也就是使高解像完全具有文字幕的功能，由鍵盤按入指令時能直接顯示在畫面上，用PRINT指令時也可以直接印字出來，當然圖形文字的混合更是方便，如此使得APPLE具有如2 — 0節所提到的其他電腦之功能，而同時又保有其本身的優點。

首先要做的是將字形製造出來，由於每字有8條掃描綫，其

中最頂綫爲了與上一行字分隔因而不畫出來（該位址存入 $00 ），譬如像 " A B C " 這些字的 8 個位元組爲：

	A	B	C
第一綫	0 0	0 0	0 0
第二綫	0 8	1 E	1 C
第三綫	1 4	2 2	2 2
第四綫	2 2	2 2	0 2
第五綫	2 2	1 E	0 2
第六綫	3 E	2 2	0 2
第七綫	2 2	2 2	2 2
第八綫	2 2	1 E	1 C

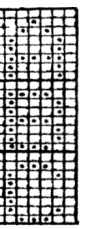


圖 2 — 4

請注意，由於每一位元與螢幕位置的關係是第一位元在最左邊，因此將每一綫上的點換成 16 進位數時必須倒過來算才行。另外，字形是以 5 × 7 的點陣來排列，因此水平綫上的 7 點最左與最右點不點亮（存入 " 0 " ），所以像 " B " 的最低綫看起來是 " ● ● ● ● ○ " 5 點其實是 " ○ ● ● ● ● ○ ○ " 7 點，另外 8 位元中所最高位元是作彩色選擇用的因此不顯示出來，此處設爲 0 ，因此 " B " 的最底綫對邊的位元組是 " ○ ● ● ● ● ○ ○ ○ " = $ 1E （由右向左算）。現在將所有文數字、符號等的字形資料列出如下：

```
*B000.B1FF

B000- 00 1C 22 2A 3A 1A 02 3C   @
B008- 00 08 14 22 22 3E 22 22   A
B010- 00 1E 22 22 1E 22 22 1E   B
B018- 00 1C 22 02 02 02 22 1C   C
B020- 00 1E 22 22 22 22 22 1E   D
B028- 00 3E 02 02 1E 02 02 3E   E
B030- 00 3E 02 02 1E 02 02 02   F
B038- 00 3C 02 02 02 32 22 3C   G
B040- 00 22 22 22 3E 22 22 22   H
B048- 00 1C 08 08 08 08 08 1C   I
B050- 00 20 20 20 20 20 22 1C   J
B058- 00 22 12 0A 06 0A 12 22   K
B060- 00 02 02 02 02 02 02 3E   L
B068- 00 22 36 2A 2A 22 22 22   M
B070- 00 22 22 26 2A 32 22 22   N
B078- 00 1C 22 22 22 22 22 1C   O
B080- 00 1E 22 22 1E 02 02 02   P
B088- 00 1C 22 22 22 2A 12 2C   Q
B090- 00 1E 22 22 1E 0A 12 22   R
B098- 00 1C 22 02 1C 20 22 1C   S
B0A0- 00 3E 08 08 08 08 08 08   T
B0A8- 00 22 22 22 22 22 22 1C   U
B0B0- 00 22 22 22 22 22 14 08   V
B0B8- 00 22 22 22 2A 2A 36 22   W
B0C0- 00 22 22 14 08 14 22 22   X
B0C8- 00 22 22 14 08 08 08 08   Y
B0D0- 00 3E 20 10 08 04 02 3E   Z
B0D8- 00 3E 06 06 06 06 06 3E   [
B0E0- 00 00 02 04 0A 12 20 00   \
B0E8- 00 3E 30 30 30 30 30 3E   ]
B0F0- 00 00 00 08 14 22 00 00   ^
B0F8- 00 00 00 00 00 00 00 3E   _
B100- 00 00 00 00 00 00 00 00
B108- 00 08 08 08 08 08 00 08   !
B110- 00 14 14 14 00 00 00 00   "
```

```
B118- 00 14 14 3E 14 3E 14 14    #
B120- 00 08 3C 0A 1C 28 1E 08    $
B128- 00 06 26 10 08 04 32 30    %
B130- 00 04 0A 0A 04 2A 12 2C    &
B138- 00 08 08 08 00 00 00 00    '
B140- 00 08 04 02 02 02 04 08    (
B148- 00 08 10 20 20 20 10 08    [
B150- 00 08 2A 1C 08 1C 2A 08    \
B158- 00 00 08 08 3E 08 08 00    _
B160- 00 00 00 00 00 08 08 04    ,
B168- 00 00 00 00 3E 00 00 00    -
B170- 00 00 00 00 00 00 00 08    .
B178- 00 00 20 10 08 04 02 00    /
B180- 00 1C 22 32 2A 26 22 1C    0
B188- 00 08 0C 08 08 08 08 1C    1
B190- 00 1C 22 20 18 04 02 3E    2
B198- 00 3E 20 10 18 20 22 1C    3
B1A0- 00 10 18 14 12 3E 10 10    4
B1A8- 00 3E 02 1E 20 20 22 1C    5
B1B0- 00 38 04 02 1E 22 22 1C    6
B1B8- 00 3E 20 10 08 04 04 04    7
B1C0- 00 1C 22 22 1C 22 22 1C    8
B1C8- 00 1C 22 22 3C 20 10 0E    9
B1D0- 00 00 00 08 00 08 00 00    :
B1D8- 00 00 00 08 08 00 08 04    ;
B1E0- 00 10 08 04 02 04 08 10    <
B1E8- 00 00 00 3E 00 3E 00 00    =
B1F0- 00 04 08 10 20 10 08 04    >
B1F8- 00 1C 22 10 08 08 00 08    ?
```

表 2 — 1

總共列了 64 個字形，由於每個字須要佔用 8 個位元組，因此需要 512 個共兩頁的位址，以第三頁是無法容納得下的，因此存到 B000 至 BAFF 共兩頁的位址區間。

查表程式

最後只剩下查表的程式了，由於本程式要接受按鍵訊號（如此由鍵盤送過來的指令才能被接收），因此就以鍵盤字元碼來考慮，APPLE字形總共只有 64 個（表 2－1），因此只要取其低 6 位元便足以組合 64 個字形（$2^6 = 64$），由於（表 2－1）是依ASCIL次序排列的，第 0 個字形"@"的資料是由 $B000～$B007，第 1 個字形的資料是由 $B008～$B00F，接着是 $B014～$B017，……，由此可發現第一綫資料的位址分別是 $B000、$B008、$B010、$B018、……，也就是0→$0、1→$8、2→$10、3→$18、……剛好位址是字元碼的 8 倍，因此程式的設計很簡單，只要將接收的字元碼乘以 8 倍再加上 $B000便是對應的起始位址了。假設字元碼存在 A 暫存器內，程式設計如下：

```
TABSEAR   AND  #$3F      取低 6位元
          ASL
          ASL      }     乘8
          ASL
          STA  $0E      低位元組
          LDA  $B0
          ADC  #$00     } 高位元組
          STA  $0F
```

程式　2－2

這樣便可以找出對應的字元碼了。譬如"A"由鍵盤送過來的是$C1，AND#$3F之後變成$01，乘以$8之後變成$08所以起始位址是$B008，與(表2－1)一致。

程式整合

全部的工作都完成了，現在還剩下的是如何取得字元碼？有一個很好的想法是，所有APPLE要輸出的字元都要經由COUT，如果我們將COUT的程序稍予改變，先令COUT JUMP到高解像字形處理程序來，處理完畢後再將主權交回給APPLE，如此一方面能夠執行我們的程式，一方面不阻礙APPLE的正常功能。

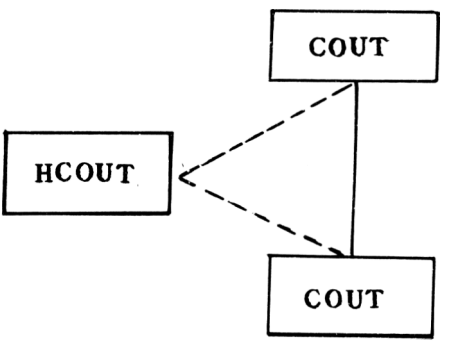


圖2－5

因爲一切已準備妥當，所以主程式的編寫很簡單，列出如下

```
$00=VERTICAL COORDINATE
$0E=TABLE POSITION-LO
$0F=TABLE POSITION-HI
$14=(HBASL)
$15=(HBASH)
       PHA
       STA $C050
       STA $C052
       STA $C057
       CMP #$8D      RETURN?
       BEQ $0349
       TABSEAR
       LDA CV
       ASL           CV*8=FIRST
       ASL           LINE VERTICAL
       ASL           COORDINATE
       STA $00       SET VERTICAL VALUE
       TXA
       PHA
       TYA
       PHA
       HSTADV  ==> HBASCALC
       PLA
       TAY
       PLA
       TAX
       PLA
       JMP COUT1
```

程式　2－3

上述程式中，利用CV的值來換成（乘以8）高解像畫面的垂直座標，由於HSTADV也使用到CH，因此HTAB及VTAB等BASIC指令是可以用來控制本程式的印字位置的，而且最後的JMP　COUT 1，更可以經由ADVANCE副程式來使CH值加一，如此巧妙的利用COUT的一些功能，使我們的HCOUT更加簡潔而方便。

現在將程式列出如下：

```
*300LLLLLL
 768
0300-   48           PHA
0301-   8D 50 C0     STA   $C050
0304-   8D 52 C0     STA   $C052
0307-   8D 57 C0     STA   $C057
030A-   C9 8D        CMP   #$8D
030C-   F0 3B        BEQ   $0349
030E-   29 3F        AND   #$3F
0310-   0A           ASL
0311-   0A           ASL
0312-   0A           ASL
0313-   85 0E        STA   $0E
0315-   A9 B0        LDA   #$B0
0317-   69 00        ADC   #$00
0319-   85 0F        STA   $0F
031B-   A5 25        LDA   $25
031D-   0A           ASL
031E-   0A           ASL
031F-   0A           ASL
0320-   85 00        STA   $00
0322-   8A           TXA
0323-   48           PHA
0324-   98           TYA
0325-   48           PHA
0326-   A2 08        LDX   #$08
0328-   A0 00        LDY   #$00
032A-   20 4D 03     JSR   $034D
032D-   18           CLC
032E-   A5 04        LDA   $04
0330-   65 24        ADC   $24
0332-   85 04        STA   $04
0334-   A5 05        LDA   $05
0336-   69 00        ADC   #$00
0338-   85 05        STA   $05
```

```
033A-   B1 0E       LDA   ($0E),Y
033C-   91 04       STA   ($04),Y
033E-   E6 00       INC   $00
0340-   E6 0E       INC   $0E
0342-   CA          DEX
0343-   D0 E5       BNE   $032A
0345-   68          PLA
0346-   A8          TAY
0347-   68          PLA
0348-   AA          TAX
0349-   68          PLA
034A-   4C F0 FD    JMP   $FDF0
034D-   A5 00       LDA   $00
034F-   0A          ASL
0350-   0A          ASL
0351-   29 1C       AND   #$1C
0353-   85 05       STA   $05
0355-   A5 00       LDA   $00
0357-   6A          ROR
0358-   6A          ROR
0359-   6A          ROR
035A-   6A          ROR
035B-   29 03       AND   #$03
035D-   05 05       ORA   $05
035F-   09 20       ORA   #$20
0361-   85 05       STA   $05
0363-   A5 00       LDA   $00
0365-   6A          ROR
0366-   29 E0       AND   #$E0
0368-   85 04       STA   $04
036A-   6A          ROR
036B-   6A          ROR
036C-   29 18       AND   #$18
036E-   05 04       ORA   $04
0370-   85 04       STA   $04
0372-   60          RTS
0373-   FF          ???
0374-   FF          ???
```

程式　2－4

仔細的將$B000～$B1FF的字形資料及$0300～$0372的程式鍵入，然後以*36:00 03或CTRL－C之後POKE 54，0:POKE 55，3就可以開始執行程式了（但是用300 G或CALL 768却不行，爲何？）。程式一開始，螢幕變成高解像畫面，由鍵盤按鍵（*或］均可）則字形出現在最底下一列，現在鍵入

```
]HTAB 20:VTAB 10:PRINT "ABC"
```

結果眞的能夠在螢幕中央印出"ABC"三個字，我們發現HTAB，VTAB，PRINT這三個指令竟然眞的能夠直接在高解像畫面上使用，目的已經達成。

由於PRINT是字元輸出的指令，每次PRINT被使用時，便會呼叫到COUT，而COUT又已經指向$0300，因此每次由PRINT輸出一字元碼（存於A暫存器），便到$0300來印在高解像畫面上，而程式最後是以JMP COUT 1來結束的，因此便又回到COUT 1來在眞正的文字幕上印字（所以RESET後，會在文字幕上看到與高解像畫面一樣的文數字），爲什麼要JMP COUT 1：主要是利用其COUT 1的ADVANCE、VTAB、等副程式，使得程式中的CH、CV能夠設定成正確值，而VTAB、HTAB等BASIC指令也才會被執行。

接着我們來看一個例子，首先將上述的程式鍵入到$0300～$0372區間，同時將表2－2的文數字形表鍵入至$B000～$B1FF，然後RESET鍵入下列程式：

```
]LIST

0  REM   *** A DEMON. OF MIXING G
      RAPHIC AND TEXT ***
10  HGR
20  POKE 54,0: POKE 55,3
25  FOR I = 0 TO 7
30  HTAB 5: VTAB 12: PRINT "*** T
      HIS IS A DEMON. PROGRAM ***"

35  FOR J = 0 TO 1000: NEXT J
40  HCOLOR= I: HPLOT 0,0
45  CALL 62454: REM   *** PAINTIN
      G THE FULL SCREEN TO COLOR *
      **
50  NEXT I
70  REM   *** MOVIES ***
75  FOR K = 1 TO 11
80  VTAB 10: HTAB 13
85  PRINT  MID$ ("MIXING !!! MIXI
      NG !!!",K,K + 9)
90  FOR D = 0 TO 100: NEXT D
95  NEXT K
100  GOTO 75
101  REM   *** NOTE! TO USE THIS
      PROGRAM YOU MUST ENTER THE M
      ACHINE LANGUAGE PROGRAM AND
      THE TABLE STATED ABOVE ***
```

程式　2－5

程式首先以HGR來清除高解像畫面，接着將（CSWL、H）指向$0300，這時$0300至$0372已存入程式2－4的機械語言程式了，然後以HTAB及VTAB指定在高解像畫面中央PRINT出文字來，第45行是呼叫變換底色副程式，當顏色變換

完畢，接着由第 75 行至 100 行來在高解像畫面顯示動態的文字幕。程式執行結果如下：

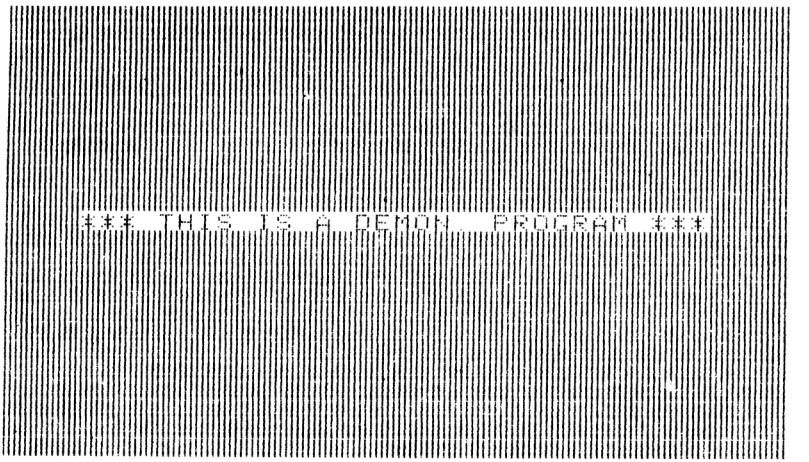


圖 2－6

現在我們已造出與小教授電腦一樣的功能了，至於螢幕上捲的功能，讀者是否也可以仿 1 — 6 節的方法來處理呢？

2-3 在高解像畫面造出低解像圖形

要造出低解像圖形，基本程式架構與文數字的情形完全一樣，唯一不同的地方便是表格的差異，因此只要附加上低解像圖形表，便可以在高解像畫面上造出低解像方塊，如此可以令程式 2 — 4 的功能更加强化了，當然查表程式 TABSEAR 也要修改才行。

但是要使用 PLOT 指令之前必須先設定 COLOR，因此C-

OLOR的設定值也要取到查表程式來用，如此才能方便的以BASIC指令操作程式。記得* COLOR資料存放位址$30吧?只要下達BASIC的COLOR指令，該顏色的編碼值便存入到$30，因此只要在COLOR指令之後來取$30的值便可以取到顏色編碼了。同樣的PLOT X，Y的座標值是分別存放於$F0及$2D（或$2C），因此只要在PLOT X，Y之後來取$F0及$2D（或$2C）的值，便可以找出正確的座標位置了。

所以現在只要設計一個程式，使得在COLOR=N：PLOT X，Y之後能夠到$30及（$F0，$2D）去取得資料，然後由TABSEAR查表，由HSTADV印出圖形便可以了。不巧的是，PRINT指令是呼叫COUT，所以可以利用（CSWL、H）來指向程式的開頭而使程式自動執行，並且最後以JMP COUT 1來將控制權交回APPLE，但是PLOT指令並沒有如此的指位器，因而程式無法自動執行，唯一的方法是以CALL來完成，但是這樣對於使用上並不增加什麼困擾，只不過是在正常的程式內增加一個CALL指令而已，譬如：

```
]LIST

10   HGR
20   COLOR= 9
30   PLOT X,Y
40   CALL 892
50 ........
60 ........
```

程式 2－6

* 第一冊P.37

上列程式中，第10行是因爲要在高解像畫圖，因此使用HGR，第20行及第30行是正常的低解像顏色及畫圖指令，在此我們是利用這兩個指令來設定$30，$F0及$2D的內含值，第40 行便是叫出畫圖副程式的指令。

現在要做的便是造出查表副程式及方塊圖形的表格，我們將圖形表格接在字形表格之後，也就是由$B200開始的位址，如果依照字形方式來安排表格，則必須佔用16色×8條掃描綫 = 128位元組。但是高解像畫面只有8色*，如此便可以將記憶位址減半，而低解像方塊每一條掃描綫都是同一格式，如果使用HSTVADV填資料時，每次取同一數據格式，那麼只要8個元組便足夠使用。現在將8個顏色所對應的位元組列出如下：

```
JCALL-151

*B200:00 2A 55 7F 80 AA D5 FF
```

至於查表程式，首先須由$30取來顏色碼，由第一册2－3節**知道顏色碼的高半位元組是與低半位元組存同樣的資料，因此只取低半位元組便可以。低半位元組共$2^4 = 16$色，但是現在只需要8色，所以要先除以2，然後再進行查表的工作。現在將查表程式列出如下：

---

* 依照APPLE的定義爲：黑1、綠、紫、白1、黑2、橙、藍、白2。

** P 37、38

```
*
TABSEAR   LDA  $30
          AND  #$0F
          LSR  A
          STA  TABINDEX
          .........
          .........
```

程式　2－7

整個程式架構與程式 2 － 4 類似，其中HBA SCALC 是完全一樣的，如果與程式 2 － 4 合併起來的話就更方便，其中HBASCALC 可以由二者來共用，現在將完整的程式列出如下：

```
037C-   48        PHA
037D-   A5 30     LDA   $30
037F-   29 0F     AND   #$0F
0381-   4A        LSR
0382-   85 02     STA   $02
0384-   A9 B2     LDA   #$B2
0386-   85 03     STA   $03
0388-   A5 2D     LDA   $2D
038A-   EA        NOP
038B-   0A        ASL
038C-   0A        ASL
038D-   0A        ASL
038E-   85 00     STA   $00
```

```
0390-   8A          TXA
0391-   48          PHA
0392-   98          TYA
0393-   48          PHA
0394-   A2 08       LDX   #$08
0396-   A0 00       LDY   #$00
0398-   20 4D 03    JSR   $034D
039B-   18          CLC
039C-   A5 04       LDA   $04
039E-   65 F0       ADC   $F0
03A0-   85 04       STA   $04
03A2-   A5 05       LDA   $05
03A4-   69 00       ADC   #$00
03A6-   85 05       STA   $05
03A8-   B1 02       LDA   ($02),Y
03AA-   91 04       STA   ($04),Y
03AC-   E6 00       INC   $00
03AE-   CA          DEX
03AF-   D0 E7       BNE   $0398
03B1-   68          PLA
03B2-   A8          TAY
03B3-   68          PLA
03B4-   AA          TAX
03B5-   68          PLA
03B6-   60          RTS
```

程式　2－8

將程式 2－4 也一併由磁帶輸入，如此便可以使高解像畫面具有畫高解像圖形、低解像方塊及印出文數字形的功能。下例便是一個混合三種功能的示範程式：

```
]LIST

10  HGR
30  FOR I = 0 TO 23
40  COLOR= I / 3
45  PLOT I,I: CALL 892
46  PLOT 39 - I,I: CALL 892
50  NEXT I
60  POKE 54,0: POKE 55,3: VTAB 10
     : HTAB 7: PRINT "'THREE IN O
    NE' DEMON PROGRAM"
65  HPLOT 20,20 TO 259,20 TO 259,
    191 TO 20,191 TO 20,20
70 A = 1
80  GOTO 70
```

程式　2－9

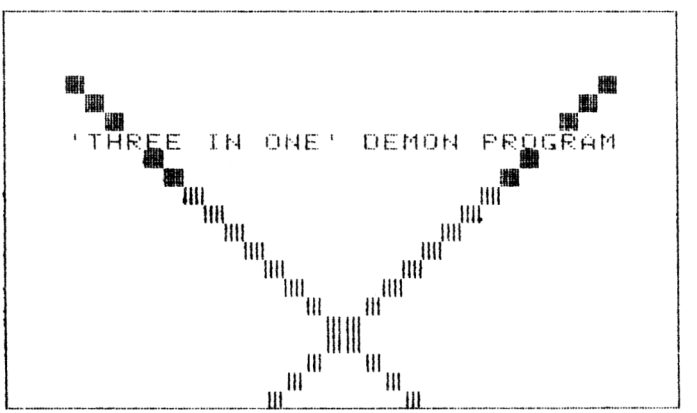
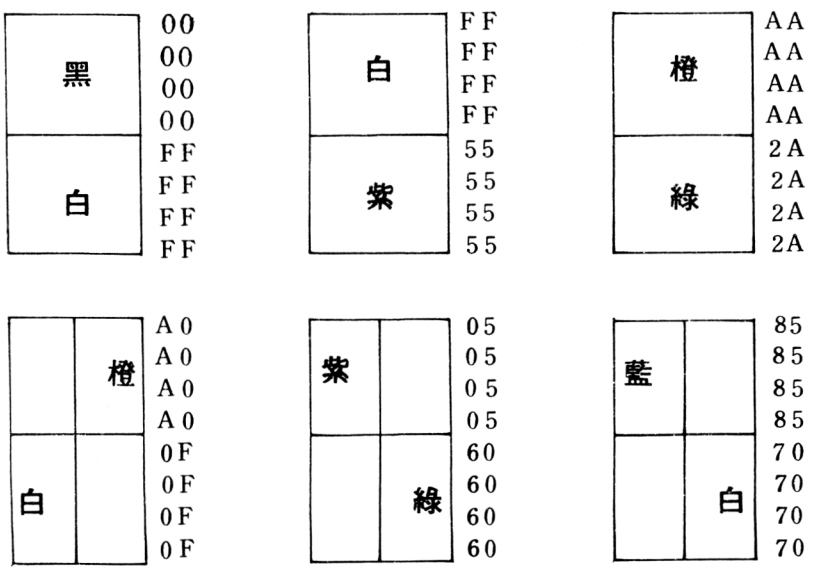


圖 2－7

在這個例子裡，我們可以看出，除了每個PLOT指令後必須加上CALL 892的指令外，其餘幾乎與正常BASIC無異，對於需要使用圖形的場合，可說是有相當的幫助。

由於我們是以8條掃描綫來畫出一個方塊，因此畫出來的低解像方格是比原來的低解像畫面高度增加一倍，也就是說畫面只分成40×24個方塊。如果將$394的LDX #$08改成LDX #$04，或是仿照APPLE PLOT副程式（$F800）的方式，利用MASK可以造出40×48的畫面格式。在圖2－7中，是文字與低解像方塊混合然後以高解像細綫劃框。

如果將本章所述的方法加以應用，只要在$B208之後附加上一些圖形單元的圖形表，便可以利用SHIFT或CTRL鍵來造出各種不同的圖形單元，下圖便是一些簡單的圖形：



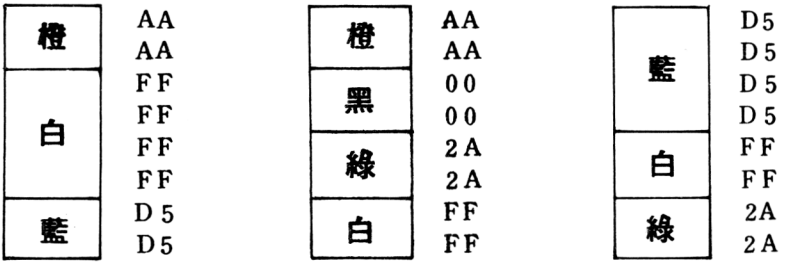


圖 2 — 8

至於像"♥ ♠ 日 月"等圖形單元，可以依照 2 — 1 節所述的原則及下一章的查表技巧，當作一番練習吧！將程式 2 — 4 、程式 2 — 8 以及相關圖形表錄到卡帶上，對於圖表描繪有很大的幫助。

```
]LIST

0  REM  *** GRAPHIC ELEMENTS IN HI-RES ***
1  HGR
5  FOR I = 0 TO 7: READ W(I): NEXT I: REM  *
     ** READ GRAPHIC DATA ***
10  VTAB 22: INPUT "KEY IN (X,Y)";X,Y
20  DEF  FN D(Y) =  INT (Y / 8)
30  DEF  FN M(Y) = Y -  FN D(Y) * 8
35  FOR J = 0 TO 7:A = 8192 + X +  FN D( FN
     D(Y)) * 40 +  FN M(Y) * 1024 +  FN M( FN
     D(Y)) * 128: POKE A,W(J):Y = Y + 1: NEXT
     J
40  GOTO 5
800  DATA 0,20,62,62,28,8,0,0,0,8,28,62,54,8
     ,28,0,0,62,34,62,34,62,0,0,0,62,34,62,3
     4,62,34,49
```

創新的BASIC指令 3

& TINY INTEPRETER

∞蘋 果 開 花∞

或許我們會這樣想：除了小教授這種與APPLE特性相近的電腦以外，APPLE是否也可以模擬其它類型電腦的特殊機能？？另外，對於一些常常使用CALL呼叫的副程式，是否可以賦予BASIC的指令名稱？譬如：喇叭發聲副程式可以在BASIC中使用SOUND指令來執行，CTRL—G可以使用BELL指令來呼叫、……等等，凡此皆可以使程式解讀容易明白。還有一些有用的指令，譬如APPLE Ⅲ的SWAP、INTEGER BASIC的GOTO A、GOSUB A……　所有以上提到的功能，在本章將會一一介紹並附加到APPLESOFT BASIC上，使APPLE功能更加强化。

要將這些功能組合在一個程式裡，並且由BASIC指令來呼

叫使用，等於是編寫了一個小型的解譯程式一樣，因此本章另一個重要的目的便是學習來造出一個小型的解譯程式（ TINY INTEPRETER ）。

3—0 "&"與TXTPTR

" & " 這個符號在英文中常用來代替AND這個字，它的發音爲" AMPERSAND "。

APPLE有一個手册上未列出的特性，那便是一旦在BASIC程式中遇到以" & "開頭的指令，便會跳到$3F5去執行程式。如果在$3F5寫下一段程式，便可以利用&來呼叫使用了，但是$3F5之後並沒有空下太多可以使用的位址，因此通常是在$3F5存入 JMP ××××使程式轉移到其他可使用的位址區間。

譬如，在$3F5存入 JMP $FBDD，然後鍵入：

```
]LIST

10  REM  *** DEMON PROGRAM ***
20  & : REM  *** BELL ROUTINE ***

30  GOTO 20
```

程式 3—0 BELL—DEMON

便可以使喇叭不斷的發聲，其中$FBDD是監督程式中喇叭發聲的副程式，由於這個副程式最後以RTS結束，因此& 執行完畢後又會回到BASIC來執行第30行的指令。

TXTPTR

APPLE 零頁有一對重要的指位器，稱爲 TXTPTR（TEXT POINTER ）＝（ $B8 , $B9 ），當執行 BASIC 程式 APPLE 要讀取字元或指令時，便是利用 TXTPTR 來指出所將要讀取的字元碼或指令代碼的位址。由程式 3 — 0 可以看出表面上 & 指令與 CALL 並無多大差異，對於程式解讀與說明上並無多大幫助，那麼如果在 & 之後加上文字譬如是 & BELL ，應該可以增加程式可讀性吧！要 APPLE 來讀取並辨識 & 之後的文字，就要利用 TXTPTR 這對指位器來指出每個文字的字元碼或指令代碼的位址，然後 APPLE 才到該位址上加以讀取，接下來我們將會看到 TXTPTR 如何使用。

3-1 SWAP—APPLE Ⅲ？

APPLE Ⅲ 有一個很有用的指令叫 SWAP，這個指令是用來將兩個變數的值交換，如果 A ＝ 1 ， B ＝ 2 則 SWAP ＞ A ， B 之後 ， A ＝ 2 ， B ＝ 1 。 SWAP 指令在做資料大小排列等應用時相當方便。

在 APPLE Ⅱ 上要如何來做到 SWAP 的功能呢？要完成 SWAP 這個副程式之前，必須先介紹一些零頁指位器及副程式，下表便是 3 個必須用到的零頁位址：

零 頁 位 址	
VARNAM	（$81，$82）
VARPNT	（$83，$84）
FORPNT	（$85，$86）

表 3－0

VARNAM（$81，$82）：變數名稱暫存器，存放目前使用的（LAST USED）變數名稱，由於APPLESOFT只承認變數名稱的兩個字母因此使用兩個位址來存放便夠用了。

VARPNT（$83，$84）：變數指位器，存放目前所使用的變數存在位址，通常由PTRGET找出變數位址*，位址低位元組置於A——暫存器而高位元組置於Y——暫存器，然後再以STA及STY分別存入（$83，$84）。

FORPNT（$85，$86）：一般用途指位器，此處用來存放第二個變數位址。

現在再將程式裡用到的幾個APPLE的副程式列出如下：

---

* 後述

副　程　式	
PTRGET	$DFE3
CHKCOM	$DEBE
MMCH	$DD76

表　3－1

PTRGET（$DFE3）：尋找變數位址副程式，此副程式執行時會將TXTPTR所指的變數之值存放的位址找出，並依高低位元組之順序存於Y及A——暫存器，並且存入VARPNT指位器內。

CHKCOM（$DEBE）：檢查目前TXTPTR所指的字元是否爲"，"號。

MMCH（$DD76）：印出MISMATCH ERR副程式。

現在將程式列出如下：

```
*A082LL

A082-   20 E3 DF    JSR   $DFE3
A085-   85 85       STA   $85
A087-   84 86       STY   $86
A089-   A5 81       LDA   $81
A08B-   48          PHA
A08C-   A5 82       LDA   $82
A08E-   48          PHA
A08F-   20 BE DE    JSR   $DEBE
A092-   20 E3 DF    JSR   $DFE3
A095-   68          PLA
A096-   45 82       EOR   $82
A098-   30 1F       BMI   $A0B9
A09A-   68          PLA
```

```
A09B-   45 81      EOR   $81
A09D-   30 1A      BMI   $A0B9
A09F-   A0 02      LDY   #$02
A0A1-   24 81      BIT   $81
A0A3-   30 10      BMI   $A0B5
A0A5-   24 82      BIT   $82
A0A7-   30 02      BMI   $A0AB
A0A9-   A0 04      LDY   #$04
A0AB-   B1 85      LDA   ($85),Y
A0AD-   48         PHA
A0AE-   B1 83      LDA   ($83),Y
A0B0-   91 85      STA   ($85),Y
A0B2-   68         PLA
A0B3-   91 83      STA   ($83),Y
A0B5-   88         DEY
A0B6-   10 F3      BPL   $A0AB
A0B8-   60         RTS
A0B9-   4C 76 DD   JMP   $DD76
```

程式　3－1　SWAP

這個程式執行時，遇到& 這個字之後便會到 $3F5 找出程式的進入點 $A082 ，此時 TXTPTR 已指向 " &A ，B "這個指令的第二字元 " A " 了。程式首先以 JSR $DFE3 到 PTRGET 來找出第一個變數 " A " 的值存放之位址，接着以 STA 、STY 存入 FORPNT 指位器，同時將變數名稱存於堆疊器內，然後以 JSR $DFE3 尋找 " ，" 號，" ，" 號之後就是第二個變數 " B " 了。對於第二個變數同樣以 PTRGET 找出其變數值存放之位址並且存入 VARPNT 內，然後比較 A 、B 兩個變數的類型是否相同，若不同便以 JMP $DD76 指出錯誤，接着檢知所欲交換資料的長度，最後才將 FORPNT 內的第一變數值與 VARPNT 內的第二變數值交換。

將程式 3 — 0 鍵入，並且在 $3F5存入 JMP $A082（即 4C

82 A0 ），RESET之後鍵入下列程式：

```
]LIST

10  INPUT "*** KEY IN TWO NUMBRS
     ***";A,B
20  & A,B
30  PRINT "A=";A,"B=";B


]RUN
*** KEY IN TWO NUMBRS ***100,200
A=200            B=100
```

程式　3－2　SWAP－DEMON

結果，我們發現變數A，B的值已經被“ & A,B ”這個指令交換了。至目前爲止，我們已經造出 SWAP 指令的功能，但指令的型式却是“ &　A，B ”而不是“ SWAP ＞A，B ”，要如何來改良呢？請繼續……。

3－2　GOTO A 與 GOSUB A

使用過 INTEGER BASIC的人，一定會知道GOTO及GOSUB的“目的行號”可以用變數或運算式來表示，譬如GOTO A＊2＋1或GOSUB （A／8） MOD 3 等等，這種方式對於“多條件分支”的情況，使用起來相當方便。但是在APPLESOFT 的指令當中，並不允許在GOTO或GOSUB的目的行號使用變數，雖然APPLESOFT BASIC另外以ON A GOTO及

ON A GOSUB 來代替變數行號的功能，但是對於太多目的行號的場合，是相當不方便的，讀者可以參考第一册第五章例 5 — 3 的程式第 1032 及第 1033 行便可明白。

現在，我們以一段機械語言副程式來達成 GOTO A 的功能，並透過 " & " 的方式在 BASIC 程式中來使用：

```
*A0D8L

A0D8-   20 7B DD    JSR   $DD7B
A0DB-   20 52 E7    JSR   $E752
A0DE-   20 1A D6    JSR   $D61A
A0E1-   90 03       BCC   $A0E6
A0E3-   4C 41 D9    JMP   $D941
A0E6-   A2 5A       LDX   #$5A
A0E8-   4C 12 D4    JMP   $D412
```

程式　3 — 3　GOTO A

副　程　式	
FRMEVL	$DD7B
GETADR	$E752
FNDLIN	$D61A
GOTOSP	$D941
PRROR	$D412

表　3 — 2

程式 3 — 2 使用到許多副程式，逐一說明如下：

FRMEVL（ $DD7B ）：公式計算副程式，這個副程式將TXT-

PTR所指的公式加以計算並將結果存入FAC零頁暫存器。由於使用了FRM-EVL，才能計算GOTO目的行號的公式，譬如：GOTO A＊2＋1。

GETADR（ $E752 ）：行號計算副程式，這個副程式將FAC所存的數值（ ± 65535 ）*，轉換成兩位元組的正整數（ 0 — 65535 ），並存入LINNUM指位器。LINNUM是零頁的行號指位器。

FNDLIN（ $D61A ）：程式尋找副程式，找出由LINNUM 所指行號的程式所在位址。

GOTOSP（ $D941 ）：特殊的GOTO副程式，將TXTPTR指向“由FNDLIN找出的程式”之第一個字元碼或指令代碼。

ERROR（ $D412 ）：錯誤檢知副程式。

明白了上述的副程式之後，程式3 — 2也就不難加以解讀了。首先由JSR FRMEVL來計算GOTO之後的算式（譬如A＊2＋1），並將結果存入FAC，接着由GETADR副程式來將FAC的值換成行號值並存入LINNUM指位器，然後以FNDLIN來找出程式所在，如果找到程式則以GOTO SP來將程式轉換（ TXTPTR指向）到目的行號。如果在FNDLIN執行時，找不到目的行號，此時進位旗號C = 0，則由ERROR來印出UNDEFIND LINE NUMBER STATEMENT。

將程式3 — 2鍵入，並將$3F5指向程式進入點$A0D8，並

* 參考第一册 附錄B之FAC

鍵入下列BASIC 程式：

```
]LIST

10  REM  *** DEMON PROGRAM FOR GO
     TO A ***
20  INPUT "*** KEY IN A NUMBER BE
     TWEEN 0-99 ***";A
30  &  INT (A / 50) * 10 + 40
40  PRINT A;" IS LESS THAN 50"
45  GOTO 20
50  PRINT A;" IS GREATER OR EQUAL
      TO 50"
60  GOTO 20

]RUN
*** KEY IN A NUMBER BETWEEN 0-99 ***10
10 IS LESS THAN 50
*** KEY IN A NUMBER BETWEEN 0-99 ***90
90 IS GREATER OR EQUAL TO 50
```

程式　3－4　　GOTO A－DEMON

對於GOSUB 這個指令，也可以類似GOTO的方式做出程式來，現在列出如下：

```
*A0EBL

A0EB-   20 7B DD   JSR   $DD7B
A0EE-   20 52 E7   JSR   $E752
A0F1-   20 1A D6   JSR   $D61A
A0F4-   90 1D      BCC   $A113
A0F6-   A9 03      LDA   #$03
A0F8-   20 D6 D3   JSR   $D3D6
A0FB-   A5 B9      LDA   $B9
A0FD-   48         PHA
A0FE-   A5 B8      LDA   $B8
A100-   48         PHA
A101-   A5 76      LDA   $76
A103-   48         PHA
A104-   A5 75      LDA   $75
A106-   48         PHA
A107-   A9 B0      LDA   #$B0
A109-   48         PHA
A10A-   20 B7 00   JSR   $00B7
A10D-   20 41 D9   JSR   $D941
A110-   4C D2 D7   JMP   $D7D2
A113-   A2 5A      LDX   #$5A
A115-   4C 12 D4   JMP   $D412
A118-   FF        ???
```

程式　3－5　　GOSUB A

我們也類似GOTO A的方式寫一個程式來試驗GOSUB A：

```
]LIST

10   INPUT "*** KEY IN A NUMBER BE
      TWEEN 0-99 ***";A
20   &  INT (A / 50) * 10 + 30
25   GOTO 10
30   PRINT "*** A IS LESS THAN 50
      ***"
33   PRINT
35   RETURN
40   PRINT "*** A IS GREATER OR EQ
      UAL TO 50 ***
43   PRINT
45   RETURN
```

程式　3－6　　GOSUB A－DEMON

3-3 SOUND、BELL及其他

除了上幾節所介紹的指令外，其他一些常用的機械語言副程式也可以拿來當成&指令，譬如下列程式便是一個常用的發聲副程式：

```
*A0BCL

A0BC-   20 46 E7    JSR   $E746
A0BF-   86 51       STX   $51
A0C1-   A4 50       LDY   $50
A0C3-   98          TYA
A0C4-   AA          TAX
A0C5-   CA          DEX
A0C6-   D0 FD       BNE   $A0C5
A0C8-   2C 30 C0    BIT   $C030
A0CB-   45 51       EOR   $51
A0CD-   AA          TAX
A0CE-   CA          DEX
A0CF-   D0 FD       BNE   $A0CE
A0D1-   2C 30 C0    BIT   $C030
A0D4-   88          DEY
A0D5-   D0 EC       BNE   $A0C3
A0D7-   60          RTS
```

程式　3－7　　SOUND

將程式3－6鍵入之後，將$3F5指向$A0BC，下列便是一個驗證的程式，程式中A，B是SOUND的音高及音長的參數，改變不同的A、B數值，便可以造出不同的音效。

```
]LIST

10 A = 0:B = 0
20  & A,B
30 A = A + 10:B = B + 10
40  IF A < 255 AND B < 255 THEN 2
    0
50  END
```

程式　3－8　　SOUND－DEMON

至於一些APPLE內部具有的副程式，只要將進入點存入$3F 5～7的位址內，便可以直接由&指令來使用了。下表便是一些常用的APPLE內藏副程式：

副 程 式	
BELL	$FBDD
CTEOP	$FC42
CTCOL	$F3F6

表 3—3

BELL（$FBDD）：喇叭發出單音（IKHZ）副程式，存在於監督程式內。

CTEOP（$FC42）：清除游標以下畫面副程式，存在於監督程式內。

CTCOL（$F3F6）：底色設定副程式，清除高解像畫面成爲彩色底色，底色顏色由上一次畫點的顏色來決定，這個副程式存在於BASIC解譯器之內。

3—4 創新的BASIC指令

雖然我們已經建立許多可以由&指令來呼叫的副程式，但是這與使用CALL呼叫並無不同，並不能顯示出其優點。現在我們要寫一個程式，這個程式能夠給上述的每一副程式一個BASIC指令名稱，而使得使用上如BASIC指令一般方便。這些指令是：

& SOUND A, B	由喇叭發出長短高低不同的聲音
& SWAP > A, B	將 A，B 兩變數之數值交換
& GOTO A	可以使用變數行號的 GOTO
& GOSUB A	可以使用變數行號的 GOSUB
& CLEAR TO COL	將高解像畫面塗上底色
& CLEAR TO EOP	將文字幕從游標以下部份清除令
& BELL	喇叭發出 BELL 的聲音

表　3 — 4

要達成指令辨認的工作，使得表 3 — 4 中的每個指公下達之後，可以呼叫到各別的副程式來使用，最重要的便是造出一個查表核對程式，當查出某一指令名稱之後，便跳至其對應的副程式工作，若是查不出相同的名稱，便印出？SYNTAX ERROR，如此便與 APPLE 原來的 BASIC 指令具有完全的功能了。至於要如何由 BASIC 程式中來呼叫查表程式，只要像表 3 — 4 一樣的在每個指令之前加上 " & " 符號便可以了。

下列便是查表的程式，其中零頁位址 $09 是用來存放指令的排列順序，$DEC9 副程式是印出？SYNTAX ERROR 的副程式，$D998 是將 Y 暫存器的內含值加入到 TXTPTR 的副程式，這個副程式使得主程式得以逐一檢查指令的每一字元或指令代碼（TOKEN）是否正確。

```
A013-   A2 00       LDX   #$00
A015-   86 09       STX   $09       指令名順序位置
A017-   A0 00       LDY   #$00
A019-   BD 48 A0    LDA   $A048,X   取出一個字元碼
A01C-   F0 1A       BEQ   $A038     指令結束？
A01E-   C9 FF       CMP   #$FF      所有指令名均
A020-   F0 08       BEQ   $A02A     核對過了？
A022-   D1 B8       CMP   ($B8),Y   比較字元碼或
A024-   D0 07       BNE   $A02D     代碼是否相同
A026-   C8          INY             取下一個字元
A027-   E8          INX             碼或代碼來比較
A028-   D0 EF       BNE   $A019
A02A-   4C C9 DE    JMP   $DEC9     若全不相同
                                    印出ERROR
A02D-   E8          INX
A02E-   BD 48 A0    LDA   $A048,X   取下一個指令
A031-   D0 FA       BNE   $A02D
A033-   E8          INX
A034-   E6 09       INC   $09       設定指令順序
A036-   D0 DF       BNE   $A017
A038-   20 98 D9    JSR   $D998     取下一個來比較
A03B-   06 09       ASL   $09
A03D-   A6 09       LDX   $09
A03F-   BD 75 A0    LDA   $A075,X
A042-   48          PHA
A043-   BD 74 A0    LDA   $A074,X
A046-   48          PHA
A047-   60          RTS
```

程式　3－9　　查表程式

程式3－9，首先將零頁位址$09設定成0，也就是由表格的第一個指令開始查對，表格是存放在$A048到$A069的位址區間內，每個指令名稱之間以00做爲間格，因此當查到00之後便可以知道是一個指令名稱的終止。至於指令所對應的副程式的進入位址是存放在$A074至$A081之間，但是這裡存的是進入位址減一而不是進入點本身。查表是以X暫存器做爲索引核對來

比較&之後的指令與表格內存的指令，BASIC 程式的指令 是 以 TXTPTR 來指位的，TXTPTR 是（ $B8，$B9 ），所以比較的手續是在 $A022 以 CMP （ $B8 ），Y 執行的，一旦找出相同的指令，便以 $09 所指指令位置存入堆疊位址區內，最後當 RTS 時，程式便直接轉移到堆疊所存位址加一的位址，此位址恰好是指令對應副程式的進入點這也便是爲什麼在 $A074 至 $A081 之間所存的是位址減一的原因了。在表 3－5 的指令名稱，因爲是 APPLESOFT BASIC 所以字元碼是使用正 ASCII 方式* 而譬如 CLEAR 、TO 等 BASIC 原有的指令則以指令代碼表示，這是必須注意的，譬如 CLEAR TO COL 的編碼爲 $BD(CLEAR)，$C1(TO)， $43(C)，$4F(O)，$4C(L)。

```
*A048.A06F

A048- AB 00 B0 00 53 4F 55 4E
A050- 44 00 42 45 4C 4C 00 BD
A058- C1 43 4F 4C 00 BD C1 45
A060- 4F 50 00 53 57 41 50 CF
A068- 00 FF FF FF FF FF FF FF
```

表 3－5 指令名稱表

```
*A074.A081

A074- D7 A0 EA A0
A078- BB A0 DC FB F5 F3 41 FC
A080- 81 A0
```

表 3－6 指令副程式進入位址表

最後，將完整的程式列出如下，這個程式具有表 3－4 所列

的 7 種新增指令，程本身是由主程式 3 — 9 附加上程式 3 — 1 、3 — 3 、3 — 5 、3 — 7 以及表 3 — 3 所列的副程式構成的，由 $A000 至 $A012 這段程式是用來設定& 向量、還原 HIMEM 及字串指位器等初值化工作，以免程式被字串資料覆蓋過去了，因此 BASIC 程式最初應該以 CALL 40960 來完成 $A000 的工作。如果是在使用磁碟機的場合，則必須注意到將程式移到與 DOS不相重疊的區域。本程式共由 279 個位元組構成，結構簡單，如果能夠加以發揮（修改表 3 — 5 ，表 3 — 6 ），應該可以無限發展的。仿此您是否也可以造出 CALL SOUND、CALL XPLOT 、CALL YPLOT 、……？

```
*A000LLLLLLLL

A000-   A2 02      LDX   #$02
A002-   BD 10 A0   LDA   $A010,X
A005-   9D F5 03   STA   $03F5,X
A008-   95 4F      STA   $4F,X
A00A-   CA         DEX
A00B-   10 F5      BPL   $A002
A00D-   4C 8C F2   JMP   $F28C
A010-   4C 13 A0   JMP   $A013
A013-   A2 00      LDX   #$00
A015-   86 09      STX   $09
A017-   A0 00      LDY   #$00
A019-   BD 48 A0   LDA   $A048,X
A01C-   F0 1A      BEQ   $A038
A01E-   C9 FF      CMP   #$FF
A020-   F0 08      BEQ   $A02A
A022-   D1 B8      CMP   ($B8),Y
A024-   D0 07      BNE   $A02D
A026-   C8         INY
A027-   E8         INX
A028-   D0 EF      BNE   $A019
A02A-   4C C9 DE   JMP   $DEC9
A02D-   E8         INX
A02E-   BD 48 A0   LDA   $A048,X
A031-   D0 FA      BNE   $A02D
A033-   E8         INX
A034-   E6 09      INC   $09
A036-   D0 DF      BNE   $A017
A038-   20 98 D9   JSR   $D998
A03B-   06 09      ASL   $09
A03D-   A6 09      LDX   $09
A03F-   BD 75 A0   LDA   $A075,X
A042-   48        PHA
A043-   BD 74 A0   LDA   $A074,X
```

```
A046-   48          PHA
A047-   60          RTS
A048-   AB          ???
A049-   00          BRK
A04A-   B0 00       BCS   $A04C
A04C-   53          ???
A04D-   4F          ???
A04E-   55 4E       EOR   $4E,X
A050-   44          ???
A051-   00          BRK
A052-   42          ???
A053-   45 4C       EOR   $4C
A055-   4C 00 BD    JMP   $BD00
A058-   C1 43       CMP   ($43,X)
A05A-   4F          ???
A05B-   4C 00 BD    JMP   $BD00
A05E-   C1 45       CMP   ($45,X)
A060-   4F          ???
A061-   50 00       BVC   $A063
A063-   53          ???
A064-   57          ???
A065-   41 50       EOR   ($50,X)
A067-   CF          ???
A068-   00          BRK
A069-   FF          ???
A06A-   FF          ???
A06B-   FF          ???
A06C-   FF          ???
A06D-   FF          ???
A06E-   FF          ???
A06F-   FF          ???
A070-   FF          ???
A071-   FF          ???
A072-   FF          ???
A073-   FF          ???
A074-   D7          ???
```

```
A075-   A0 EA       LDY   #$EA
A077-   A0 BB       LDY   #$BB
A079-   A0 DC       LDY   #$DC
A07B-   FB          ???
A07C-   F5 F3       SBC   $F3,X
A07E-   41 FC       EOR   ($FC,X)
A080-   81 A0       STA   ($A0,X)
A082-   20 E3 DF    JSR   $DFE3
A085-   85 85       STA   $85
A087-   84 86       STY   $86
A089-   A5 81       LDA   $81
A08B-   48          PHA
A08C-   A5 82       LDA   $82
A08E-   48          PHA
A08F-   20 BE DE    JSR   $DEBE
A092-   20 E3 DF    JSR   $DFE3
A095-   68          PLA
A096-   45 82       EOR   $82
A098-   30 1F       BMI   $A0B9
A09A-   68          PLA
A09B-   45 81       EOR   $81
A09D-   30 1A       BMI   $A0B9
A09F-   A0 02       LDY   #$02
A0A1-   24 81       BIT   $81
A0A3-   30 10       BMI   $A0B5
A0A5-   24 82       BIT   $82
A0A7-   30 02       BMI   $A0AB
A0A9-   A0 04       LDY   #$04
A0AB-   B1 85       LDA   ($85),Y
A0AD-   48          PHA
A0AE-   B1 83       LDA   ($83),Y
A0B0-   91 85       STA   ($85),Y
A0B2-   68          PLA
A0B3-   91 83       STA   ($83),Y
A0B5-   88          DEY
A0B6-   10 F3       BPL   $A0AB
A0B8-   60          RTS
A0B9-   4C 76 DD    JMP   $DD76
A0BC-   20 46 E7    JSR   $E746
A0BF-   86 51       STX   $51
A0C1-   A4 50       LDY   $50
```

```
A0C3-  98          TYA
A0C4-  AA          TAX
A0C5-  CA          DEX
A0C6-  D0 FD       BNE   $A0C5
A0C8-  2C 30 C0    BIT   $C030
A0CB-  45 51       EOR   $51
A0CD-  AA          TAX
A0CE-  CA          DEX
A0CF-  D0 FD       BNE   $A0CE
A0D1-  2C 30 C0    BIT   $C030
A0D4-  88          DEY
A0D5-  D0 EC       BNE   $A0C3
A0D7-  60          RTS
A0D8-  20 7B DD    JSR   $DD7B
A0DB-  20 52 E7    JSR   $E752
A0DE-  20 1A D6    JSR   $D61A
A0E1-  90 03       BCC   $A0E6
A0E3-  4C 41 D9    JMP   $D941
A0E6-  A2 5A       LDX   #$5A
A0E8-  4C 12 D4    JMP   $D412
A0EB-  20 7B DD    JSR   $DD7B
A0EE-  20 52 E7    JSR   $E752
A0F1-  20 1A D6    JSR   $D61A
A0F4-  90 1D       BCC   $A113
A0F6-  A9 03       LDA   #$03
A0F8-  20 D6 D3    JSR   $D3D6
A0FB-  A5 B9       LDA   $B9
A0FD-  48          PHA
A0FE-  A5 B8       LDA   $B8
A100-  49 A5       EOR   #$A5
A102-  76 48       ROR   $48,X
A104-  A5 75       LDA   $75
A106-  48          PHA
A107-  A9 B0       LDA   #$B0
A109-  48          PHA
A10A-  20 B7 00    JSR   $00B7
A10D-  20 41 D9    JSR   $D941
A110-  4C D2 D7    JMP   $D7D2
A113-  A2 5A       LDX   #$5A
A115-  4C 12 D4    JMP   $D412
```

程式　3 — 10

```
]LIST

10  REM    *** DEMON PROGRAM FOR 7
      '&' COMMANDS ***
20 A = 100:B = 200
30  PRINT "A=";A,"B=";B
40  & SOUNDA,B
50  & SWAP > A,B
60  GOSUB 250: REM  *** TIME DELA
    Y ***
70  PRINT : PRINT "A=";A,"B=";B
80  & SOUNDA,B
'90  &  GOTO A
99  &  GOSUB B + 1
100  & BELL: END
101  GR : TEXT
110  VTAB 1: INVERSE : PRINT "CLE
    AR TO END OF PAGE"
115  GOSUB 250
120  &  CLEAR  TO EOP
130  GOSUB 250
140  RETURN
200  HGR : HCOLOR= 2
210  HPLOT 0,0
220  &  CLEAR  TO COL
230  GOSUB 250
240  &  GOTO A - 101
250  FOR I = 0 TO 1000: NEXT I
260  RETURN
300  REM    *** "& SOUND A,B" ==>
     "&SWAP > A,B" ==> "& GOTO A"
      ==> "& CLEAR TO COL" ==> "&
      GOSUB B" ==> "& CLEAR TO EO
     P" ==> "& BELL" ***
```

程式　3－11　'&'－DEMON

WRITE READ 徹底研究 4

本章將告訴你，APPLE 如何將資料寫出至磁帶，如何由磁帶讀取資料。如果明白其中原理，READ 、WRITE 對你將不再是個謎了……

微電腦是一種軟體與硬體很接近的機器。在設計上，如果硬體能力充份的話，將給軟體設計者很大的方便；如果軟體設計巧妙的話，硬體優異的功能才得以完全發揮。同樣的，在學習上，如果要徹底瞭解任何一部微電腦的動作細節，則非對硬體、軟體同時解析不爲功。

4—0 硬體線路：$C020與$C060

根據APPLE參考手册第98頁(英文版)所述，APPLE 將 $C000 至 $C07F 共128個位址的範圍，交由綫路板上位置編號爲

F13 的74 LS138 IC 來分配各個位址的工作。這128 個位址，以非完全解碼的方式，被 74 LS138 分解 8 等份，所謂完全解碼是指解碼器的每一輸出綫，唯一對應到 $0000至$FFFF中唯一的一個位址，而非完全解碼，則是如表 4 — 1 的大部份位址所示的，每一輸出綫對應到一個以上的位址。

A0	A1	A2	Z0	Z1	Z2	Z3	Z4	Z5	Z6	Z7
0	0	0	0	1	1	1	1	1	1	1
1	0	0	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1
1	1	0	1	1	1	0	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0

表　4 — 0　74LS138　解碼眞值表

A0	A1	A2	位　　址　　功　　能
0	0	0	$C000 ～ $C00F　鍵盤　　— 16384
1	0	0	$C010 ～ $C01F　鍵盤旗號 — 16368
0	1	0	$C020 ～ $C02F　卡帶輸出 — 16352
1	1	0	$C030 ～ $C03F　喇叭　　— 16336
0	0	1	$C040 ～ $C04F　GAME　I／O　PIN5
1	0	1	$C050 ～ $C05F　螢幕開關及 GAME I／O
0	1	1	$C060 ～ $C06F　卡帶輸入及 GAME　I／O
1	1	1	$C070 ～ $C07F　GAME　控制桿

表　4 — 1　APPLE 內置 I／O 功能配置圖

表4 — 1之中，與我們有關的是卡帶輸入及輸出位址，其硬體相關綫路見圖4 — 0。

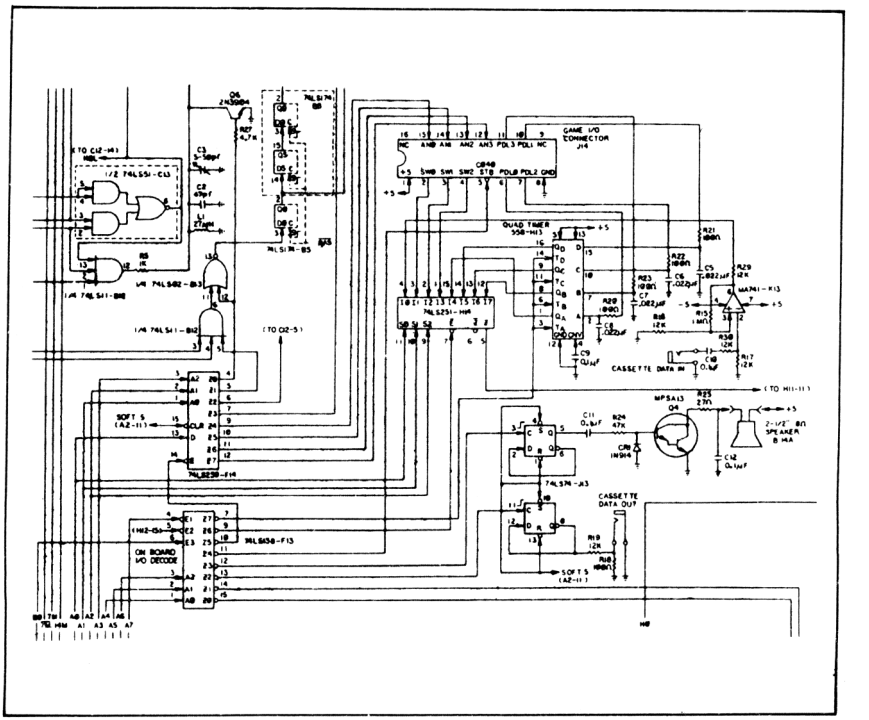


圖　4 — 0 APPLE內部I／O解碼綫路

由圖4 — 0的F13 — 74LS138可以看出其3條解碼選擇輸入綫A0、A1、A2是由位址綫A4、A5、A6來選取，而位址綫A7則輸入至IC選用端E1，由此可知A7必須是低電位時（邏輯0），才可能起用74LS138，所以如果假設74LS138的輸出綫位址爲$IJKL，則K必然在0～7之間，這是$A_7$爲零的緣故，至於

IJ 則是由74LS138 的另一個選用端 E2 來決定的，它是由位置編號爲H12 的另一個 74LS138 的第15接脚 Z0 綫所控制，這條 Z0 綫的對應位址爲 $C0××（非完全解碼），所以F13的輸出綫位址是 $C0KL，其中K爲0 至 7 的數，L 則爲 0 至 F 的數，亦即 $C000 ～ $C07F 。

與卡帶輸出入位址相關的是 K = 2,6 的情況，當 K = 2 時 Z2 被選上，Z2 的訊號接往位置編號爲J13的74LS74 的 CLOCK 輸入端，74LS74 是內含兩個 D 型正向觸發的正反器，兩個正反器其中一個是用來觸動喇叭，另一個是輸出往卡式錄音機。兩個正反器均將 $\overline{Q}$ 輸出端接往資料輸入端 D，因此構成爲 T 型正反器*，只要 CLOCK 輸入端有一正向脈衝輸入，Q 輸出馬上反轉狀態，但是脈衝負向下降時並不能觸發正反器，所以每一個 CLOCK 只能令正反器反轉一次狀態，必須兩次脈衝觸發才能在輸出端造出一個脈衝。這種特性，使得 APPLE 可以利用選擇位址的時間間隔來控制 J13 輸出脈衝的週期。

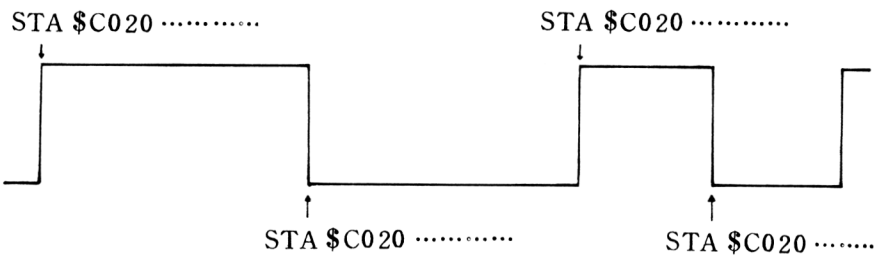


圖 4－1

當 K = 6 時，F13 的輸出綫 Z6 被選上，因此觸發 H14 的第 7 脚 $\overline{E}$，如果這時位址綫 A0、A1、A2 亦同時爲 0，則 H14 的輸

* 見Millman 微電子學習題 7－11 。

入綫 I0 被選上（完全解碼），這時卡式輸入訊號經由K13 的 741 整形後，便經由H14 的 I0 而由第 5 脚的 Z 送往 H11 的第11 脚，H11 的第 11 脚與第 9 脚並聯爲 CPU 的 D7 資料綫，因此當以 LDA $C060 由卡帶輸入端讀取資料時，卡帶的訊號電平便經由 H14 送到資料綫的 D7 而由 CPU 讀取。以上便是 APPLE 卡帶資料讀寫的硬體動作，但是如何以這種硬體動作來表示軟體內容呢？

4—1 串列訊號的處理

由上一節，我們可以看出卡帶資料傳送的路綫只有一條，也就是正反器的 Q 輸出或資料綫的 D7，這是典型的串列訊號輸送的例子。串列資料是一整串的 0 與 1 ，也就是說要將每一位元組所包含並排的 8 個位元前後依次一個個的來傳送，以典型的非同步串列資料傳送方法來說，這些資料的傳送必須有一定的格式，如此發射設備與接收設備才能依此共同認定的格式來辨認訊號內容。譬如，在一長串的訊號當中，如何辨別每一位元組的起始與終止，這便須要制定一套規則來遵守，才能使彼此的語言加以溝通，這也便是不同電腦之間的卡帶程式不能通用的原因之一了（當然其系統、位址結構、語言也不同）。

圖 4 — 2 便是一種常用的串列訊號格式，這種方法，利用一個 0 訊號做爲起始位元（當做字元碼開始的指示），而用兩個 1 訊號做爲終止位元（當做該字元碼結束的指示），這種方式使得一個 7 位元 ASCII 碼的傳送必須使用 10 個位元的訊號來處理（ 7 + 1 + 2 = 10 ）。如果再加上極性核對位元（ Parity Check

）則須 11 位元。

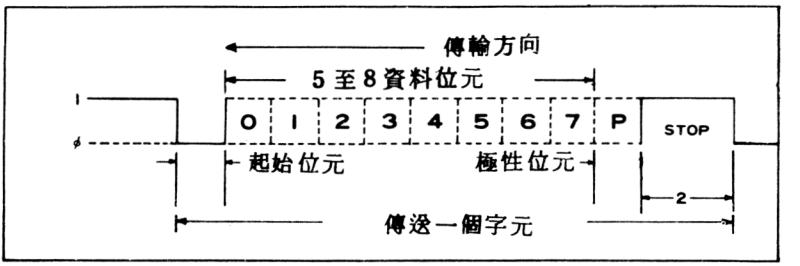


圖　4 — 2

要造出圖 4 — 2 的串列資料，可以依下列動作來完成：

1. 由輸出端送出一起始位元（ 0訊號 ）。
2. 將欲送出之位元組取入 A 暫存器。
3. 以 LSR 的方式將每一位元由最低位元依次移入進位旗號（ 因爲低位元在先 ），並將進位旗號的內含值送往輸出端。
4. 送出極性位元及終止位元。

至於要讀取圖 4 — 2 的資料，則可依下列方式爲之：

1. 檢知起始位元。
2. 延遲一位元時間（ 因爲眞正的資料在起始位元之後 ）。
3. 讀取一位元資料進入 D7 ，以 ROR 的方式移入低位元（ 因爲低位元在先 ）。
4. 計數至資料位元結束。
5. 讀取極性位元來判斷，並辨別資料終止位元。

對於直接傳送的資料，譬如TTY等，上述的方式便可使用，但是磁帶訊號是經過錄音帶及各種電子設備的傳送、放大及衰減，在這些過程當中，可能會發生訊號變形及相位變化，可能原來爲1的訊號變成0而0變成1。爲了避免這種現象，對於0與1的訊號就不能以電位的高、低爲依據了，以目前微電腦所採用的絕大多數是使用所謂的“FSK”方式，這種稱爲Frequency Shift keying的方式，故名思義可以知道是以調頻的方式來處理的，也就是以不同的頻率（週期）來代表0與1的訊號，如此移相、反相就再也不是問題了，不過一定要以全週期或其整數倍來代表0與1，否則一經反相便又不知如何分辨了。

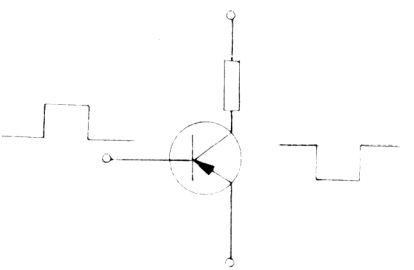


圖　4－3

4-2　常用的卡帶讀寫方式

本節介紹幾種常用的卡帶資料讀寫方式，請注意，此處我們只强調其位元及位元組格式，至於其他如檔案名稱（Filename）格式，則只略提及而已。

㈠K.C.S 方式

一般的手提卡式錄音機很適合廣泛的被使用來做爲微電腦的資料儲存，爲了訂定出一套統一的卡式錄音帶資料儲存方式，於是許多微電腦廠家於1975年11月在Missouri 州的 kansas市開會決定一套大家同意的卡帶資料存取的標準方法，這個方法就稱爲 " KANSAS CITY " * 標準，或 KCS 方式。

KCS 方式主要的原理是認爲一般的卡帶錄音機無法將一個數位的方波完整的予以重現，因此資料的讀取可能發生錯誤。爲了解決這個問題，於是將本來以電位高低來判別 0 與 1 訊號的調幅方式，改成以頻率高低來判別 0 與 1 訊號的調頻方式，而且只用了兩種頻率 1200HZ 及 2400HZ 。這種調頻的方式稱爲 FSK （ Frequency Shift Keying ）方式，其資料傳送速度爲 300Bd ，而且可以容忍卡帶速度正負 25 %的變化。KCS 的基本規範如下：

1. 連續 8 週期的 2400 HZ 訊號代表一個 " 1 " 位元。
2. 連續 4 週期的 1200 HZ 訊號代表一個 " 0 " 位元。
3. 每一個字元訊號以一個 " 0 " 做爲起始指示位元，並且以 2 個或以上 " 1 " 做終止指示，在起始與終止指示訊號之間，就是 8 個位元的資料訊號。
4. 位元組之間的時間間隔不予限定。
5. 字元的位元的串列順序是以最低位元開始，最高位元結束，之後可再加（或不加）一個校對位元（ Parity ），但是連校對位元一併算入，總共不可超過 8 個位元。
6. 如果採用的位元總數少於 8 個，那麼剩下的位元區間要以高位

* Peschke, M. and V. Peschke " BYTE'S Audio Cassette Standards Symposium " BYTE, NO. 6, Feb. 1976, P.72.

元（ 8 個 2400 HZ 表示一個高位元 ）來填滿。

7. 資料長度不限制，但是要以 2400 HZ 來產生 5 秒鐘的前導訊號。

下圖是 K.C.S. 訊號波形的說明：

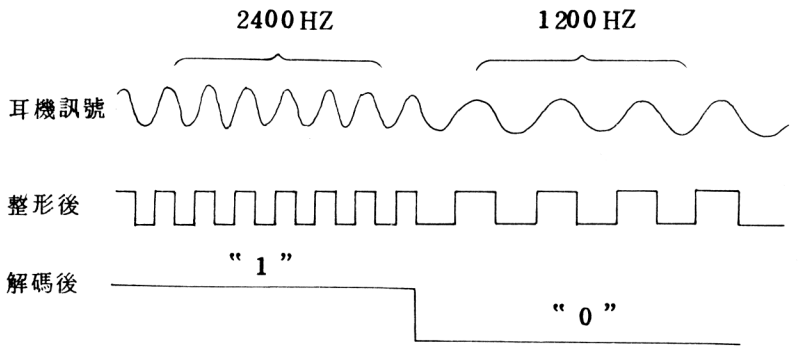


圖 4—4 KCS 訊號波形說明圖

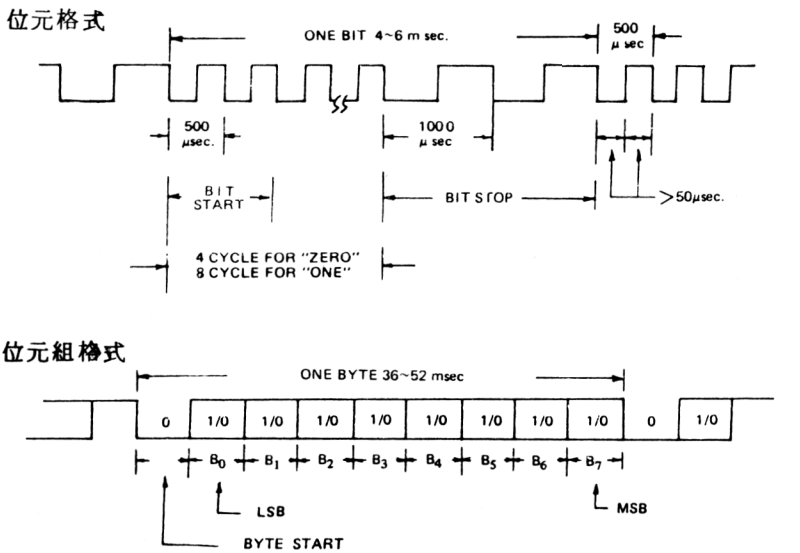
K.C.S. 方式的讀寫速率約爲 300 Baud 。使用 KCS 的機種較多，如 Ohio Scientfic 的 C1P , Motorola 的 6802D5 等皆採用此方式。

㈡ EDU — 80 方式

EDU — 80 爲早期國內廣爲使用的微電腦學習機，其磁帶錄音格式爲：

1. 每一位元由 2 個 0.5 msec 的方波作爲起始指示，接着以 4 個週期的方波代表資料 " 0 "，以 8 個週期的方波代表資料 " 1 "，最後由兩個 1 msec 的方波做爲位元終止指示。

2. 每一位元組以一個 " 0 " 位元做爲起始指示，接着爲 8 個位元的資料，排列次序以低位元在前，高位元在後。
3. 以連續15秒鐘的 1 msec 方波做爲資料存至卡帶時的前導波，而讀取卡帶則必須有 5 秒以上的 1 msec 前導訊號才 被承認爲正確訊號。
4. 前導波之後以 2 個0.5 msec 的方波指示正式資料開始。
5. 正式資料以10個位元組做開始，來指出程式名稱，起始位址，終止位址、程式長度及核對位元組，接下來的才是程式本身。
6. 由於採用上述的錄音格式，因此資料讀寫速率約爲 22.7 byte／sec ≈ 200 Baud 。



4 — 5　EDU 80 卡帶訊號說明

㈢AIM — 65　方式

1.位元格式：每個位元以2400 HZ的正半週訊號做爲起始指示，其後跟着3個半週的訊號，如果此3個半週之頻率爲2400 HZ則代表" 1 "訊號（邏輯1），若爲1200 HZ則代表" 0 "訊號（邏輯0）。

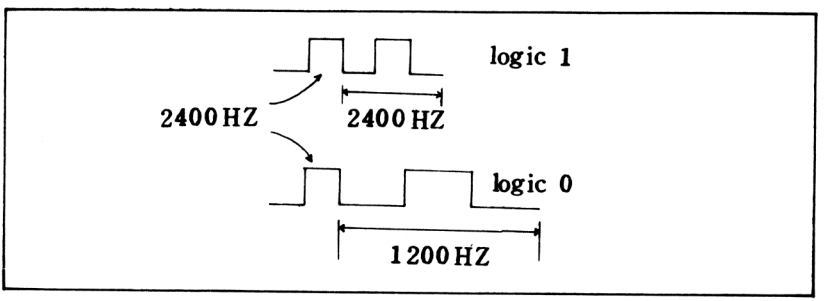


圖　4 — 6

2.位元組格式：每一位元組的8個位元以最高位元（MSB）首先被送出及接收。

3.資料以分段（Block）的方式來存放，每段以36個同步位元組做前導，其後爲格式識別及段數指示位元組，接着才是79個資料位元組，最後以兩個核對（Checksum）位元組做結束。

㈣RCA VIP — 1802方式

1.位元格式：以一週期的2 KHZ方波代表" 0 "訊號（邏輯0），以一週期的0.8 KHZ方波代表" 1 "訊號（邏輯1）。

2. 位元組格式：以一個“ 1 ”訊號做前導，接下來的才是依低位元至高位元順序的8 個位元資料，最後再加上一個極性（Parity）核對位元。
3. 以 4 秒鐘 2KHZ 的“ 0 ”訊號爲前導訊號。

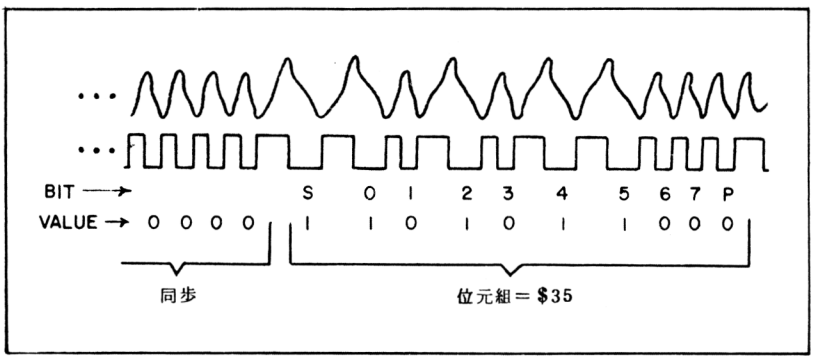


圖　4 — 7

㈤ Commodore PET　方式

1. 卡帶訊號包含 3 種頻率： 1488HZ、1953HZ、2840HZ ，分別稱爲長、中、短週期。
2. 位元格式：以一個中週期後面跟着一個短週期代表“ 1 ”訊號以一個短週期後面跟着一個中週期代表“ 0 ”訊號。基本上這是 F.S.K. 的改進。
3. 位元組格式：以一個長週期後面跟着一個中週期做爲位元組起始指示，每一位元組以一個極性核對位元做結束。
4. 前導訊號爲10秒鐘，除了做爲操作及機械緩衝之外，系統本身卽以此來做頻率參考的依據，如此可以免除錄音機速度不準的

影響。

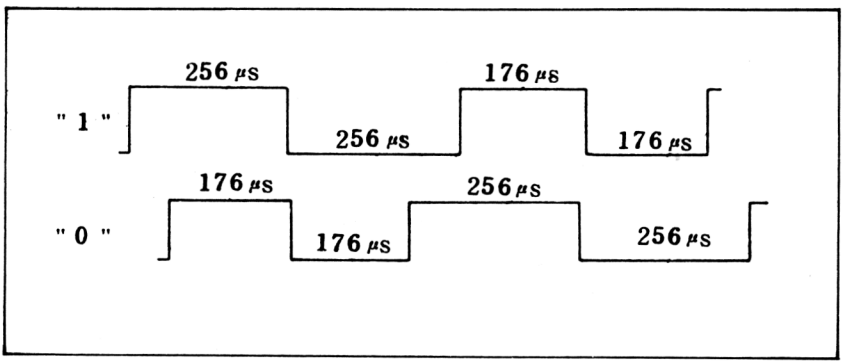


圖　4 — 8

4—3 APPLE 的 WRITE

介紹了 4 — 2 節的幾種磁帶資料讀寫方式之後，現在來看看 APPLE 的讀寫方式，爲了徹底研究其基本運作 ，我們由監督程式來做直接的探討，同時希望在瞭解其原理之後，大家可以嘗試着改變其讀寫方式，並參考 4 — 2 節的各種方式造出自己的卡帶程式來。

WRITE = $FECD

```
FECD: A9 40       490 WRITE   LDA #$40
FECF: 20 C9 FC    491         JSR HEADR
FED2: A0 27       492         LDY #$27
FED4: A2 00       493 WR1     LDX #$00
FED6: 41 3C       494         EOR (A1L,X)
FED8: 48          495         PHA
FED9: A1 3C       496         LDA (A1L,X)
FEDB: 20 ED FE    497         JSR WRBYTE
FEDE: 20 BA FC    498         JSR NXTA1
FEE1: A0 1D       499         LDY #$1D
FEE3: 68          500         PLA
FEE4: 90 EE       501         BCC WR1
FEE6: A0 22       502         LDY #$22
FEE8: 20 ED FE    503         JSR WRBYTE
FEEB: F0 4D       504         BEQ BELL
FEED: A2 10       505 WRBYTE  LDX #$10
FEEF: 0A          506 WRBYT2  ASL A
FEF0: 20 D6 FC    507         JSR WRBIT
FEF3: D0 FA       508         BNE WRBYT2
FEF5: 60          509         RTS
```

程 式　4 — 0　WRITE

程式 4 — 0 是 APPLE 監督程式中的WRITE 程式，這個程式中使用了幾個副程式，列表如下：

副　程　式	
HEADR	$FCC9
NXTA1	$FCBA
WRBIT	$FCD6
BELL	$FF3A
WRBYTE	$FEED
WRBYT2	$FEFF

表　4 — 0

首先，程式將 $40 取入A暫存器，然後進入 $FCC9 的HEADR

副程式，HEADR副程式是用來產生卡帶的前導訊號，其產生的前導訊號長度爲 T = A × 256 × 650 us ，現在A的內含值爲 #$40 = 64 ，T = 64 × 256 × 650 us = 10649600 us ≈ 10.65 秒也就是APPLE廠方所宜佈的10秒前導訊號。由HEADR 回來之後A暫存器之內含值爲$FF，請鍵入下列程式便可以知道：

```
*300L

0300-   A9 01        LDA   #$01
0302-   20 C9 FC     JSR   $FCC9
0305-   20 F0 FD     JSR   $FDF0
0308-   00           BRK

*300G
```

結果執行後，螢幕出現“ ？”號，查第一册表 2 — 0 “ ？”號對應的字形碼是 $FF ，由此可知HEADR執行之後，A暫存器的內含值爲 $FF ，其理由待HEADR之說明時再解釋。

現在繼續看WRITE 程式，由HEADR將10秒鐘的前導 訊號寫出至卡帶之後，接着將Y暫存器設定成＃$27 ，X 暫存器設定成＃$00 ，同時將A與卡帶起始位器（$3C，3D)所指而的第一個位址之內含值 EOR後，接着以PHA將此結果存入堆疊器，$FED9 的 LDA（$3C，X）又將第一位址的內含值取入A ，然後至 $FEED的WRBYTE副程式將A所內含的位元組寫出至磁帶。

WRBYTE = $FEED

這是將A暫存器的內含位元組寫入磁帶的副程式。首先設定計數，將X暫存器定爲＃$10 = 16 ，然後將A的最高位元以ASL移入進位旗號內，接着JSR WRBIT副程式將進位旗號的內含值寫出至磁帶，同時將X的值減2 ，再回到WRBYTE執行，直到X之內含值爲0 ，如此便將一位元組的8個位元寫入磁帶了。所以我們可以看出眞正執行磁帶寫出動作的是WRBIT副程式，這個副程式容後詳述。

NXTA1 = $FCBA

現在再回到主程式來，由$FEDB的JSR WRBYTE回來之後，又到$FCBA的NXTA1副程式，NXTA1的作用是將起始位址指位器的內含值減以終止位址指位器的內含值，如程式運作已達終止位址，則進位旗號爲1 ，否則爲0 ，然後將起始位址指位器之內含值加1而回到主程式。主程式則依NXTA1所決定的旗號來決定是否繼續將下一個位址之內含值寫至卡帶。

```
FCBA: A5 3C      211 NXTA1   LDA A1L
FCBC: C5 3E      212         CMP A2L
FCBE: A5 3D      213         LDA A1H
FCC0: E5 3F      214         SBC A2H
FCC2: E6 3C      215         INC A1L
FCC4: D0 02      216         BNE RTS4B
FCC6: E6 3D      217         INC A1H
FCC8: 60         218 RTS4B   RTS
```

程式　4 — 1

所以現在WRITE的動作已初具輪廓，其中$FEE3 、$FED6 、$FED8是將前後位址之內含值做EOR運算以做最後核對之用

，因爲PLA後如果進位旗號爲 1 ，則 JSR $FEED將最後一次EOR的值寫出至磁帶做爲最後一個核對位元組（ CHECKSUM ），然後到 $FF3A令喇叭發聲而結束。另外 $FED2 、$FEE1 、$FEE6 的 LDY是設定訊號週期長度用的，譬如核對位元組的週期長度是由 LDY ＃$22 來設定的，後述。

4—4 HEADR

上一節的WRITE 幾個副程式當中，還有HEADR 與WRBIT未予說明，雖然HEADR是用來寫出前導訊號的副程式 ，但是其中一些副程式却扮演着 APPLE 讀寫動作的核心角色，因此必須詳予瞭解。而WRBIT因與 HEADR 有所關連，故在本節當中一併說明。

```
FCC9  A0 4B      220 HEADR   LDY #$4B
FCCB  20 DB FC   221          JSR ZERDLY
FCCE  D0 F9      222          BNE HEADR
FCD0  69 FE      223          ADC #$FE
FCD2  B0 F5      224          BCS HEADR
FCD4  A0 21      225          LDY #$21
FCD6  20 DB FC   226 WRBIT    JSR ZERDLY
FCD9  C8         227          INY
FCDA  C8         228          INY
FCDB  88         229 ZERDLY   DEY
FCDC  D0 FD      230          BNE ZERDLY
FCDE  90 05      231          BCC WRTAPE
FCE0  A0 32      232          LDY #$32
FCE2  88         233 ONEDLY   DEY
FCE3  D0 FD      234          BNE ONEDLY
FCE5  AC 20 C0   235 WRTAPE   LDY TAPEOUT
FCE8  A0 2C      236          LDY #$2C
FCEA  CA         237          DEX
FCEB  60         238          RTS
```

程式 4 — 2

程式4 — 2 當中包含幾個重要副程式，列出如下：

副　程　式	
WRTAPE	$FCE5
ONEDLY	$FCE2
ZERDLY	$FCDB
WRBIT	$FCD6

表　4 — 1

WRTAPE = $FCE5

這是最基本的資料寫出副程式，程式首先以LDY $C020 來觸動 74LS74 的輸入端令輸出端反轉狀態，然後將＃$2C的值取入Y，X值減1（記得，在WRBYTE時，我們說X值減2），然後結束。整個動作就是令正反器輸出反轉狀態，這是WRTAPE唯一的工作！也是APPLE眞正寫磁帶的動作！

ONEDLY = $FCE2

這個副程式以DEY及BNE來做延時效果，以主程式所設定的Y值來延時，然後至WRTAPE令正反器狀態反轉，這是控制"1"訊號（邏輯1）長度的副程式，它的延時長度爲T = Y *（2 + 3）+ 2 + T(WRTAPE），其中Y爲Y暫存器的內含值，T（WRTAPE）= 4 + 2 + 2 + 6 爲WRTAPE的動作時間，此處是以CPU工作時脈週期爲單位，標準APPLE時脈週期爲1.0204 μs。計算時請注意BNE指令在分支成立時是佔用3個時脈週期，不成立時是兩個時脈週期。

ZERDLY = $FCDB

這個程式也是以DEY及BNE做延時，然後由進位旗號判斷是否直接跳至WRTAPE或是經由ONEDLY，如果經由ONEDLY，因爲Y = #$32，所以時間增加了T = 2 + 50 *（2 + 3）+ 2 − 1 = 253 個週期，其中第一項是LDY #$32的時間，最後一項是BCC不成立時省下來的時間。這是控制" 0 "訊號（邏輯0）長度的副程式，但是很特別的是ZERDLY及ONEDLY並不直接是" 0 "與" 1 "的週期時間，而是當程式由ZERDLY經過ONEDLY時，時間較長，便代表" 1 "訊號；若不經由ONEDLY直接跳至WRTAPE，時間較短便代表" 0 "訊號。由此可以發現" 1 "比" 0 "多出約253 μs，但是每次WRTAPE只是改變正反器狀態一次而已，所以對一個完整的訊號週期而言" 1 "是比" 0 "長出506 μs的時間。

WRBIT = $FCD6

現在來看WRBIT便很清楚了，WRBIT是將一個" 0 "或" 1 "位元寫出至磁帶的副程式，如果WRBIT是由程式4 — 2 的HEADR來呼叫，則因爲Y值爲#$21，C = 0（見HEADR），因此總共使用時間T = 〔2 + 2 + 6 + 2 + 2 + 2 + 2 + 6 + 33 *（2 + 3）+ 2 + 3 + 4〕+〔2 + 2 + 6 + 2 + 2 + 46 *（2 + 3）+ 2 + 3 + 4〕=〔198〕+〔253〕= 451時脈週期，這便是緊接在前導訊號之後的資料起始指示位元——前半週時間比正常" 0 "還短

的“ 0 ”（半週時間～ 200 μs ）。此處T的算法是以WRTAPE爲起點開始算的。

如果WRBIT是由程式 4 — 0 的WRBYTE來呼叫使用，則WRBIT便根據WRBYTE的ASL來看進位旗號是 0 或是 1 。若是 0 則程式 4 — 2 的WRBIT便因 BCC 成立而不經由 ONEDLY 直接到WRTAPE ，若是第一次呼叫則 Y = $27 ，磁帶訊號前半週T= 249 時脈週期，而後半週因爲 Y 值爲 $2C ，因此 T = 253 時脈週期，Y 值不同是因爲前半週是經由WRITE過來，時間多用了約30個時脈週期，所以減少 Y 值（計數值）來補償。若WRBYTE ASL到進位旗號 C 的位元是 1 ，則WRBIT經由 ONEDLY ，因此多了 253 μs × 2 = 506 μs 的時間。

爲什麼WRBIT要分前半週時間與後半週時間呢？由於前面提到 0 或 1 訊號必須時一個完整週期才可以，因此WRBIT 共 觸動 $C020 兩次，這也便是爲什麼 X 減 2 的原因了。同時，由於呼叫WRBIT的主程式來源常有不同，如此時間 T 就會不定，因此便常在主程式設定不同的 Y 值來調整時間使之一致，而後半週期由於動作過程一定，所以均設定 Y = $2C 。至於每一位元組 結 束後便要經由 NXTA1 使位址加 1 ， NXTA1 共用了約 32 ～ 36 個時脈週期，連帶其他動作時間，所以 Y 值設爲 $1D 。當整段資料全部結束後，將最後一次的核對值（ CHECKSUM ）由堆疊器以PLA方式取下後，由於時間過程也有所不同，因此 Y 值調整爲 $22 。

把上述的過程整理出來是：

1. 起始指示位元的第一半週長 Y = $21 爲 200 μs ，後半週 Y = $2C ，所以爲正常的 250 μs 。
2. 第一位元組的第一位元若爲“ 0 ”則前半週 Y = $27 ，半週長

爲250 μs 。後半週及該位元組以後之每一位元若爲 " 0 " ，則 Y = $2C ，半週長爲250 μs 。

3. 第一位元組之後的每一位元組第一位元若爲 " 0 " 則前半週，因爲 JSR NXTA1 的緣故，所以 Y = $1D ，半週長爲250 μs。其後每一 " 0 " 位元則因 Y = $2C ，半週長爲250 μs 。
4. 結束核對位元組之第一位元前半週 Y 值爲 $22 ，其後 Y 值爲 $2C 。
5. 以 ASL 方式將暫存器 A 的每一位元由 MSB 開始一一移至進位旗號，然後 WRBIT 便依 C 旗號的內含值來判斷應該輸出一 個 " 1 " 的位元訊號或是 " 0 " 的位元訊號。
6. 如果是 " 1 " 位元（邏輯 1 ），則半週長多出約 250 μs。由此可知 APPLE 以全週長爲 500 μs 的方波代表 " 0 " 位元 ，以全週長爲1000 μs 的方波代表 " 1 " 位元，這便是 APPLE 廠方公佈的規格 * ，至此我們已徹底明白了。

HEADR

現在回頭來看看程式 4 — 2 的 HEADR 前導訊號又是什麼情形。由程式 4 — 2 可看出前導訊號 Y 值爲 $4B = 75 ，除最初 256 週期進位旗號爲 0 外（ 尚未進行到 ADC #FE ），其後因爲 ADC #$FE = ADD #$FF 使得 A 值減 1，進位旗號爲 1 ，所以 256 週期過後均要經過 ONEDLY。半個週期長度約爲 T = 2 + 2 + 6 + 2 + 6 + 75 (2 + 3) + 2 + 2 + 2 + 253 + 2 + 4 = 658 時脈週期 ≈ 650 μs ，因此整個週期長度符合 APPLE 廠方公佈的 1.3 msec 。當最後 A 之值爲 0 ， ADC #$FE 之後 C = 0 ，於是

* S. Wozniak System Description : The Apple II May, 1977, BYTE.

HEADR 結束，這時 A 之內含值爲＃FF ，符合 4 — 3 節的說法。而這其中又因以 DEX 使 X 值減了256次，因此整個 HEADR 共 花了 $T = A * 256 * 650\,\mu s$ 的時間。

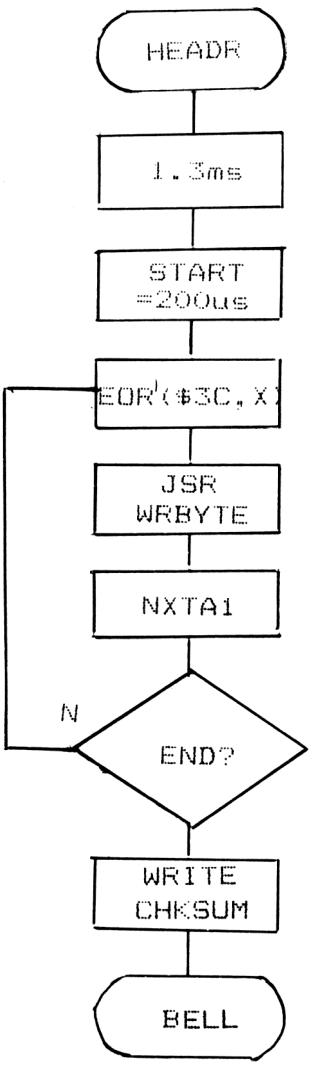

HEADR
1.3ms
START
=200us
EOR'($3C,X)
JSR
WRBYTE
NXTA1
N
END?
WRITE
CHKSUM
BELL

圖 4 — 9 A WRITE

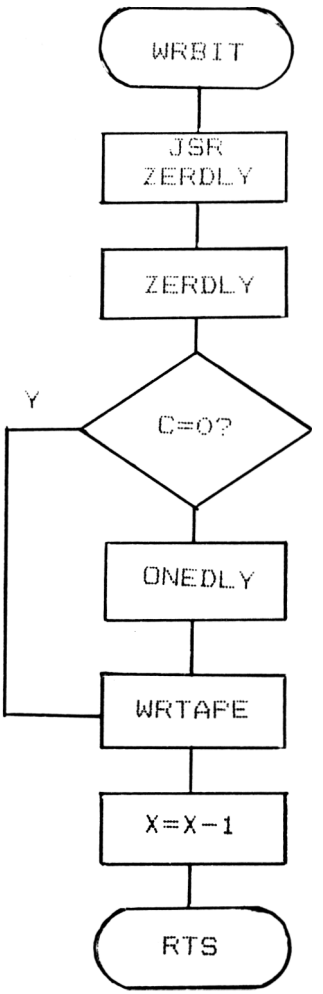


圖 4 — 9 B WRBIT

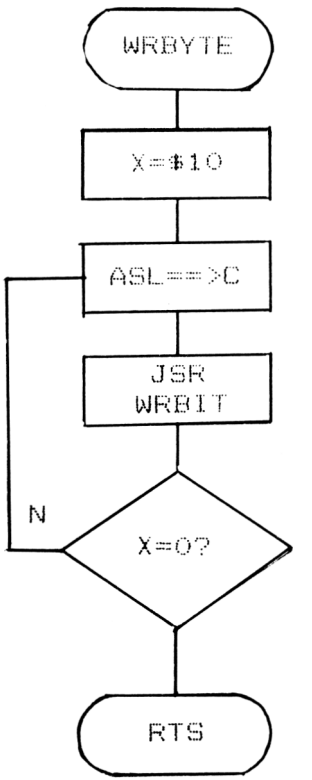


圖 4 — 9 C WRBYTE

4—5 APPLE 的 READ

瞭解WRITE各部細節動作，才有辦法解析READ這個 程式，現在再將APPLE卡帶訊號描述一遍，首先有10秒鐘週期爲1300 μs 的前導訊號，接着有一個200 μs 半週的資料起 始指示位元及隨後的250 μs 半週，再來才是眞正的資料，其中以全週期爲500 μs 的方波代表 " 0 " 位元，以全週期爲1000 μs 的方波 代表 " 1 " 位元，最後有一個核對位元組做結束，核對位元組是前後位元組 EOR運算的最後結果。

要讀回這樣的串列訊號，其過程是①檢知前導訊號②檢知資料起始位元③每 8 個位元一組的從（ $3C，$3D ）所指位址讀進每一位元組④與前一位元組 EOR 製造核對位元組⑤檢查資料是否到達（ $3E，$3F ）指示的結束位址⑥最後與卡帶最後的核對位元組比較是否正確（ 相同 ）。

```
FEFD: 20 FA FC    514 READ    JSR RD2BIT
FF00: A9 16       515         LDA #$16
FF02: 20 C9 FC    516         JSR HEADR
FF05: 85 2E       517         STA CHKSUM
FF07: 20 FA FC    518         JSR RD2BIT
FF0A: A0 24       519 RD2     LDY #$24
FF0C: 20 FD FC    520         JSR RDBIT
FF0F: B0 F9       521         BCS RD2
FF11: 20 FD FC    522         JSR RDBIT
FF14: A0 3B       523         LDY #$3B
FF16: 20 EC FC    524 RD3     JSR RDBYTE
FF19: 81 3C       525         STA (A1L,X)
FF1B: 45 2E       526         EOR CHKSUM
FF1D: 85 2E       527         STA CHKSUM
FF1F: 20 BA FC    528         JSR NXTA1
FF22: A0 35       529         LDY #$35
FF24: 90 F0       530         BCC RD3
FF26: 20 EC FC    531         JSR RDBYTE
FF29: C5 2E       532         CMP CHKSUM
FF2B: F0 0D       533         BEQ BELL
FF2D: A9 C5       534 PRERR   LDA #$C5
FF2F: 20 ED FD    535         JSR COUT
FF32: A9 D2       536         LDA #$D2
FF34: 20 ED FD    537         JSR COUT
FF37: 20 ED FD    538         JSR COUT
FF3A: A9 87       539 BELL    LDA #$87
FF3C: 4C ED FD    540         JMP COUT
```

程式　4 — 3　READ

程式4 — 3是卡帶讀取程式READ，首先將相關的副程式及零頁位址列出如下：

副　程　式	
RDBIT	$FCFD
RD2BIT	$FCFA
HEADR	$FCC9
RDBYTE	$FCEC
PRERR	$FF2D
BELL	$FF3A

表　4 — 2

零　頁　位　址	
(A1L,H)	($3C , $3D)
(A2L,H)	($3E , $3F)
CHKSUM	$2E
LASTIN	$2F

表　4 — 3

RDBIT = $FCFD

RD2BIT = $FCFA

```
FCFA: 20 FD FC    248 RD2BIT  JSR RDBIT
FCFD: 88          249 RDBIT   DEY
FCFE: AD 60 C0    250         LDA TAPEIN
FD01: 45 2F       251         EOR LASTIN
FD03: 10 F8       252         BPL RDBIT
FD05: 45 2F       253         EOR LASTIN
FD07: 85 2F       254         STA LASTIN
FD09: C0 80       255         CPY #$80
FD0B: 60          256         RTS
```

程式　4 — 4　RD2BIT

RDBIT副程式是用來將卡帶輸入的訊號（在$C060）取入A暫存器，並以EOR LASTIN（$2F）來判斷是否有電位高低（0或1）變化，如果沒變化（BPL成立），便回到RDBIT繼續判斷；如果有變化便將結果存入LASTIN，至於RD2BIT是以JSR RDBIT置於RDBIT之前來檢知兩次變化。要記得，LDA TAPEIN，是如4—0節所說的只將卡帶訊號電位狀態（0或1）取入資料綫的最高位元D7，那麼RDBIT是如何來讀取位元的呢？是否將位元讀入D7呢？不是的，我們在WRITE程式中已經知道0與1的訊號不是用電位的高低來做判斷的根據的，而是用週期長短（頻率高低）來表示的，因此必須由RDBIT中找出讀取週期長短的方法。

原來，RDBIT是以DEY及BPL來讀取磁帶訊號的電位變化時間間隔，也就是訊號半週期的讀取，所讀取的時間長度爲T＝△Y＊（2＋4＋3＋3）＋2＝△Y＊（12）＋2（算至BPL不成立爲止），其中△Y是DEY的次數，而Y的初值也如WRITE一樣的爲了調整時間長度的一致，是根據不同的呼叫RDBIT主程式來設定的。譬如，由RDBYTE呼叫時，Y值正常爲$3A，當讀取前一個位元訊號之後便回到RDBYTE，如果8個位元尚未讀完便又回到RD2BIT來，這期間共花了約40μs，然後再由RDBIT進行DEY計數，如果接下來的訊號是“0”位元，則前半週還有約210μs的持續時間（以週期爲500μs計），因此根據T＝△Y＊12＋2可算出△Y＝17＝$11，而RD2BIT在第二次執行RD2BIT又花了約22μs，因此“0”位元的後半週還有約228μs，所以△Y＝19＝$13，因此Y值由$3A減至最後還剩22＝$16，因此最後CPY #$80便使C＝0（因爲Y值小於$80

），將C值ROL進入A暫存器PHA之後便將“ 0 ”位元眞正由磁帶讀入APPLE了！如果磁帶訊號爲“ 1 ”位元，則前半週時間內△Y≈ 38 ＝ $ 26 ，而後半週△Y≈ 40 ＝ $ 28 ，因此Y由 $ 3A減了 $ 26及 $ 28後內含值變成 $ EC ，因此CPY ＃$ 80 之後C＝1 ，同樣的以ROL將C值取入A暫存器PHA後便將 “ 1 ”位元眞正的由磁帶讀入APPLE了！這便是APPLE如何以FSK方式來讀寫磁帶資料的軟體動作核心了。

如果將READ程式移到 $ 300 的RAM區間執行，請嘗式改變CPY ＃$ 80 這個指令的被運算元（ 卽＃$ 80 ），你將發現其值在＃$ 18 至＃$ EB 之間均可正確讀取程式，這是因爲CPY時Y值是 $ 16 與 $ EC只要能分辨此二數值便可以，所以照理 $ 17 至 $ EB均可用來比較， $ 17 之所以不成功是因爲我們時間計算上的誤差，而APPLE之所以採用＃$ 80 是取其平均值 ，絕不可誤解爲是爲了與D7 位元做比較才設定的。

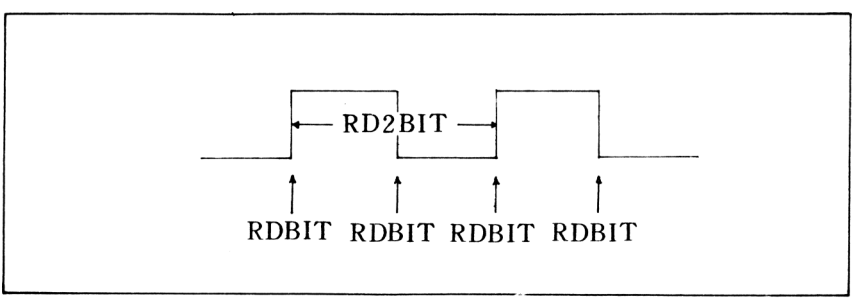


圖　4 — 10

RDBYTE = $FCEC

```
FCEC  A2 08      239 RDBYTE  LDX #$08
FCEE  48         240 RDBYT2  PHA
FCEF  20 FA FC   241         JSR RD2BIT
FCF2  68         242         PLA
FCF3  2A         243         ROL A
FCF4  A0 3A      244         LDY #$3A
FCF6  CA         245         DEX
FCF7  D0 F5      246         BNE RDBYT2
FCF9  60         247         RTS
```

程式　4 — 5　RDBYTE

這個副程式，先將X設定成位元計數器（內含值 8 ），暫存A，然後到 RD2BIT 去讀取兩次電位的變化，將結果記於進位旗號C內，接着由堆疊器取出A，以ROL A 的方式將C的值（亦即RD2BIT由卡帶讀入的值）取入A 如此繼續，直到依序由MSB爲首讀完8位元爲止。

READ

現在便可以重頭來將 READ 程式描述一遍了。程式首先以RD2BIT來檢知是否卡帶已送訊號過來，如果沒有便在 RD-2BIT中一直等待讀取 。一旦讀到有訊號變化，便設定A值爲$16（＝22 ）然後 JSR HEADR ，依照我們HEADR的公式可以算出共延時 $T = 22 \times 256 \times 650 = 3660800\,\mu s \approx 3.6$ 秒 ，這便是APPLE廠方公佈的3.5秒延時，如果磁帶前導訊號短於3.6秒，則READ副程式便因延時過長而來不及讀取資料，所以使用磁帶時不可太慢按下RETURN，在附錄中我們可以看到利用這種

特性的程式保護方式。由於HEADR之後A之值爲$FF（前述），所以STA CHKSUM便將$2E的內含值設定成$FF。

由HEADR回來之後又以RD2BIT來找是否磁帶繼續送訊號過來，若是，則設定Y值爲$24＝（36）然後到RDBIT去找接在磁帶前導訊號後的資料起始指示0半位元（即200μs的半週期），找到之後再以RDBIT跳過後半週的250μs訊號（因爲還不是要讀取的資料訊號）。接下來便開始資料的讀取了。資料的讀取以RDBYTE來進行，當讀取一位元組之後，便以STA（A1L，X）存入（$3C，$3D）所指的位址內，然後製造核對位元組，接着以NXTA1來繼續將資料指向下一位元。

當一切資料讀完，NXTA1將C設定成1，便以RDBYTE讀取置於磁帶最後面的核對位元組，並與$2E比較，如果相同則到$FF3A的BELL令喇叭發聲；最後將控制權交給COUT，如果COUT已指向程式進入點，便開始執行程式，至此我們已徹底的明白利用COUT使GAME自動執行的原理了。如果核對位元組的比較結果不同，便到FF2D的RRERR，因爲“E”的字元碼爲$C5（負ASCII，見第一册第一章），而“R”的字元碼爲$D2，所以JSR COUT之後便印出“ERR”的字形了，當然最後是以BELL結束的。當讀入GAME程式，一旦發生錯誤，由於COUT指向程式進入點，而程式又不能執行（因爲錯誤），所以畫面便停住，而游標也不能出現了（COUT已不指向COUT 1了）。

至於READ程式中許多Y值的設定，那是爲了讀取特別訊號長度（譬如200μs）或是如WRITE一般的作時間補償、設定等動作。

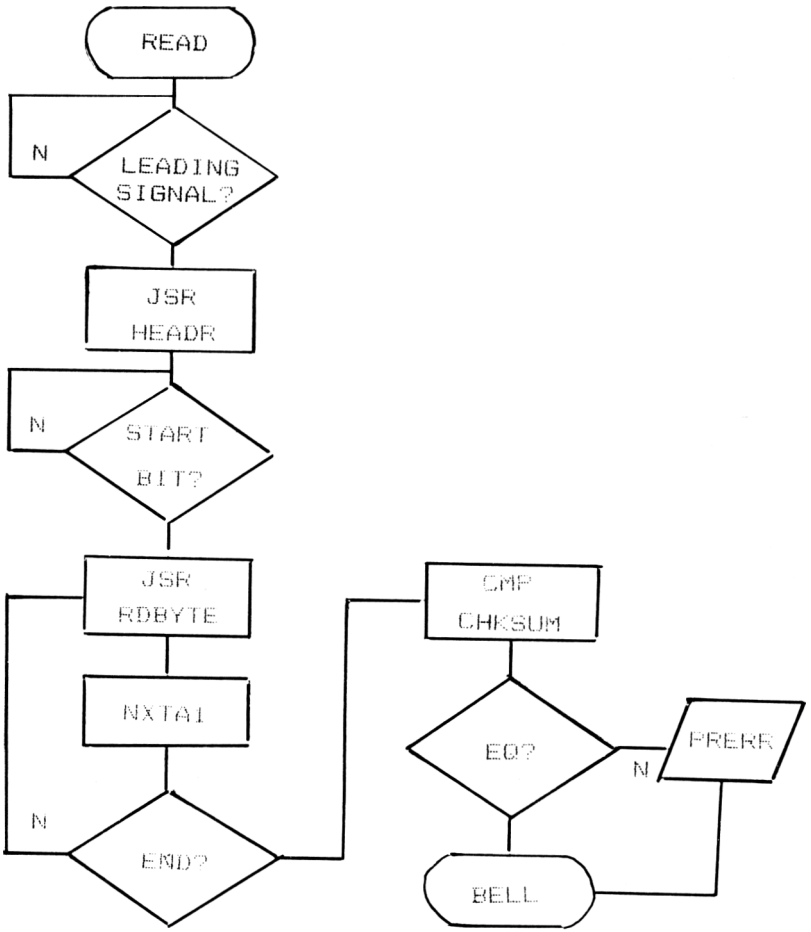
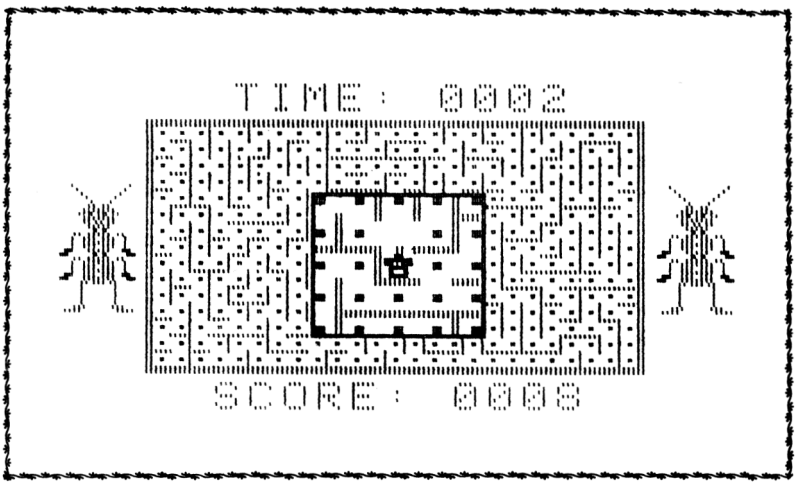

READ
N
LEADING SIGNAL?
JSR HEADR
N
START BIT?
JSR RDBYTE
NXTA1
N
END?
CMP CHKSUM
EQ?
N
PRERR
BELL

圖 4 — 11 READ

✳✳✳現在便可以輕鬆的閱讀附錄Ａ、Ｂ、Ｃ、Ｄ……✳✳✳

附錄A　金　甲　蟲

(30.300R)



A—0　30.300R

在第 0 章提到了 30・300R 這一類型的程式保護法，基本上這是利用（CSWL・H）指位器及（$3C，$3D）、（$3E，$3F）卡帶起始終止指位器來做程式的讀、寫及自行執行等功能。將一捲"金甲蟲"的程式以耳朵來聽聲音或直接讀回至 APPLE，皆可以發現這是屬於兩段式的程式，與第 0 章的"傘兵部隊"保護方式顯然有些不同，但既然是兩段程式，應該可以如 0－3 節一般的用 1030・1300R　1301・BFFFR 來讀入 APPL

E，然後以 1030．1300W　1301．BFFFW來拷貝吧？(當然拷貝的程式要再讀回來玩 GAME，則必須用 30．300R 才行，依照上述的方法來做，當第一段程式 1030．1300R 讀完之後竟然出現錯誤（ ERR ）的符號，但是不管其結果繼續照着上述的方法做下去，最後竟然也能拷貝成功！這是什麼原因呢？

A—1　程式檢討

這個程式可說是第 0 章傘兵部隊的改良現在以正規的方式，逐步來檢討細節。首先將第一段的保護程式讀到 $1030 至 $1300 的位址區間：

```
*1030.109F

1030- 00 00 FF AA 00 00 40 02
1038- 1B FD FF FF 3C 02 FF FF
1040- A9 00 85 3C A9 04 85 3D
1048- A9 00 85 3E A9 48 85 3F
1050- A9 00 85 36 A9 20 85 37
1058- A9 00 85 00 A9 04 85 01
1060- A0 00 A9 00 91 00 E6 00
1068- D0 FA E6 01 A6 01 E0 08
1070- 90 F2 AD 56 C0 AD 53 C0
1078- AD 50 C0 4C FD FE 00 00
1080- 00 00 00 00 00 00 67 7F
1088- FF FF FF FF FF FF FF FF
1090- FF FF FF FF FF FF FF FF
1098- FF FF FF FF FF FF FF FF
```

表　A—0

結果發現：

1（ CSWL、H ）是指向 $0240 。

2（ $3C ， $3D ）是指向 $023C 。

記得在第 0 章及第 4 章提到，（ $3C ， $3D ）是磁帶資料存放的記憶體起始位址，因此一旦將 $023C 存到（ $3C ， $3D ）之後，接下來的資料（即表A — 0 $103E的內含值FF ）就要由 $023E 開始了，所以由 $3E ~$023D 這段記憶位址區就被跳過不予存入資料，眞正由磁帶送來的資料是存到① $30 ～ $3D 及② $023E ～ $0300 ，這也便是爲什麼以 1030 · 1300R 來讀第一段程式會產生 ERR 的原因了（因爲（ $103C ， $103D ）並未具備（ $3C ， $3D ）那種使磁帶資料跳到 $123E 的能力，所以讀取位址長度不對而 ERR ）。

爲了證明我們的論點，請鍵入：

```
*23C:00 00 00 00
```

然後將甲蟲程式以 30 · 300R 讀入 APPLE 當喇叭發出第一個響聲時按下 RESET ，如果磁帶資料由 $3C 就開始轉移至 $023C ，則 CALL — 151 觀察 $23C 至 $23F 的內容應該是 23C ： 02 3C FF FF ，如果磁帶資料由 $3E 才開始轉移至 $023E ，則觀察到的應該是 23C ： 00 00 FF FF ，如果磁帶資料由 $40 才開始轉移至 $0240 ，則觀察到的應該是 23C ： 00 00 00 00 ，結果下表證明是如我們所推論的：由 $3E 開始轉移 * 。

---

* 由 $200 ～ $207 的 ASCII 碼是 200.27F RETURN ，詳見第一册第一章

```
*200.27F

0200- B2 B0 B0 AE B2 B7 C6 8D
0208- FF FF FF FF FF FF FF FF
0210- FF FF FF FF FF FF FF FF
0218- FF FF FF FF FF FF FF FF
0220- FF FF FF FF FF FF FF FF
0228- FF FF FF FF FF FF FF FF
0230- FF FF FF FF FF FF FF FF
0238- FF FF FF FF 00 00 FF FF
0240- A9 00 85 3C A9 04 85 3D
0248- A9 00 85 3E A9 48 85 3F
0250- A9 00 85 36 A9 20 85 37
0258- A9 00 85 00 A9 04 85 01
0260- A0 00 A9 00 91 00 E6 00
0268- D0 FA E6 01 A6 01 E0 08
0270- 90 F2 AD 56 C0 AD 53 C0
0278- AD 50 C0 4C FD FE 00 00
```

A—2 保護方式解說

既然已經知道程式是由 $3E 開始轉移，而且 COUT 也指向 $0240 ，因此接下來就查看 $023E ～ $0300 這段種式內容，其中已經知道 $23E 及 $23F 是 FF ，所以由 $240 開始列表出來：

```
*1040LL

1040-   A9 00      LDA   #$00
1042-   85 3C      STA   $3C
1044-   A9 04      LDA   #$04
1046-   85 3D      STA   $3D
1048-   A9 00      LDA   #$00
104A-   85 3E      STA   $3E
104C-   A9 48      LDA   #$48
104E-   85 3F      STA   $3F
1050-   A9 00      LDA   #$00
1052-   85 36      STA   $36
1054-   A9 20      LDA   #$20
1056-   85 37      STA   $37
1058-   A9 00      LDA   #$00
105A-   85 00      STA   $00
105C-   A9 04      LDA   #$04
105E-   85 01      STA   $01
1060-   A0 00      LDY   #$00
1062-   A9 00      LDA   #$00
1064-   91 00      STA   ($00),Y
1066-   E6 00      INC   $00
1068-   D0 FA      BNE   $1064
106A-   E6 01      INC   $01
106C-   A6 01      LDX   $01
106E-   E0 08      CPX   #$08
1070-   90 F2      BCC   $1064
1072-   AD 56 C0   LDA   $C056
1075-   AD 53 C0   LDA   $C053
1078-   AD 50 C0   LDA   $C050
107B-   4C FD FE   JMP   $FEFD
107E-   00         BRK
107F-   00         BRK
```

程式　A—0

一旦第一段保護程式讀完之後，便會利用 COUT 發聲及印出游標，但（ CSWL、H ）已指向 $240 了（亦即上列表的 $1040 ），所以便開始執行程式 A—0 的程式。

這個程式很容易的可以看出，前面部份是一連串的將磁帶資料起始及終止位址設定成 $0400 至 $4800 ，而程式進入點在 $2000 ，由 $1058 到$1071的程式是將 $0400 至 $0800的畫面區域設定成黑色（畫面黑色的對應編碼是" 00 "而文字幕空格的編碼是" A0 "），接着觸動畫面開關（ $C056 , $C053 , $C050 ）使成低解像形態，最後 JMP $FEFD（READ副程式)將接在保護程式後邊的主程式讀入到 $400 至 $4800 的區域 ，一旦讀完之後便由 COUT JMP 到 $2000 來執行金甲蟲的遊戲了。所以 .....

A－3還有一層保護

所以只要將第二段的主程式以 400．4800R 來讀入到 APPLE，然後 2000G 便可以執行遊戲了，如果要拷貝呢？ 400．4800W？結果程式好像失去了？ RESET 之後再由 2000G 開始，結果程式不能執行了！是否 $2000 之後還有保護呢？將 $2000以後的程式列出來如下：

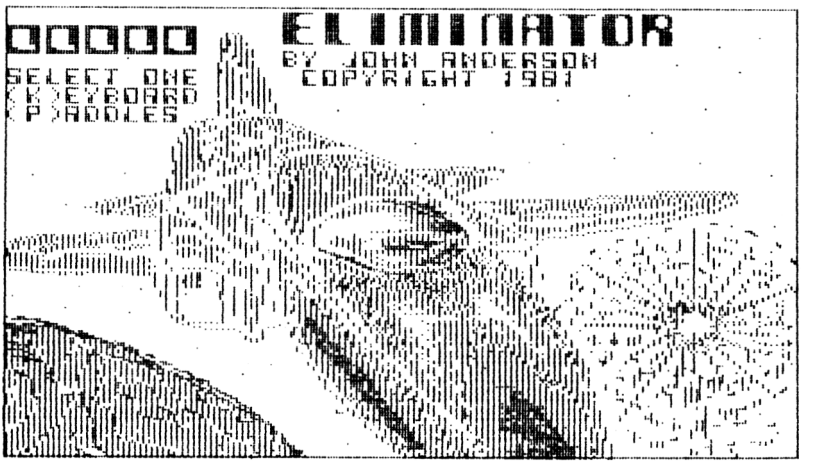
```
*2000LL

2000-   A9 00       LDA   #$00
2002-   8D 51 C0    STA   $C051
2005-   8D 54 C0    STA   $C054
2008-   AD 00 21    LDA   $2100
200B-   8D 00 80    STA   $8000
200E-   EE 09 20    INC   $2009
2011-   EE 0C 20    INC   $200C
2014-   AD 09 20    LDA   $2009
2017-   C9 00       CMP   #$00
2019-   D0 ED       BNE   $2008
201B-   EE 0A 20    INC   $200A
201E-   EE 0D 20    INC   $200D
2021-   AD 0D 20    LDA   $200D
2024-   C9 A3       CMP   #$A3
2026-   D0 E0       BNE   $2008
2028-   AD 00 44    LDA   $4400
202B-   8D 00 04    STA   $0400
202E-   EE 29 20    INC   $2029
2031-   EE 2C 20    INC   $202C
2034-   AD 29 20    LDA   $2029
2037-   C9 00       CMP   #$00
2039-   D0 ED       BNE   $2028
203B-   EE 2A 20    INC   $202A
203E-   EE 2D 20    INC   $202D
2041-   AD 2A 20    LDA   $202A
2044-   C9 48       CMP   #$48
2046-   D0 E0       BNE   $2028
2048-   A9 00       LDA   #$00
204A-   8D 10 C0    STA   $C010
204D-   AD 00 C0    LDA   $C000
2050-   C9 80       CMP   #$80
2052-   30 F9       BMI   $204D
2054-   A9 00       LDA   #$00
2056-   A2 00       LDX   #$00
2058-   A0 00       LDY   #$00
205A-   4C 00 96    JMP   $9600
```

這段程式，首先將 $2100 至 43FF 的資料搬到 $8000 至 $A

2FF 的區間，然後將 $4400 至 $47FF 的字形資料搬到 $400 至 $7FF 的區間，使顯示出片頭的說明字句，再來等待按鍵，按鍵之後便 JMP 9600 開始執行眞正的程式了。由於程式畫面用到高解像第一頁區，因此 $2000～$3FFF 的原程式便被破壞了，而再由 2000G 來執行便行不通了，正確的方法應該是 9600G。

附錄B　大　進　擊(Ⅱ)

(30.1200R)



由於＂金甲蟲＂的程式很容易的不用解析就可以拷貝，於是就拿來一捲 30.1200R 的＂大進擊（Ⅱ）＂按下 1030.2200R 2201.BFFFR，等完畢之後便以 1030.2200W 2201.BFFFW來將程式寫入至磁帶，然後 30.1200R ？……？

B—0　30.1200R

這個程式的保護方法又是上一個保護法的改進，首先還是以 1030・2200R 的方式來讀入保護程式。

```
*1030.2200R
ERR


*1030.109F

1030- 00 00 FF AA 0B 28 00 02
1038- 1B FD 00 10 3C 00 FF 0B
1040- 30 00 42 60 30 60 FF FE
1048- 00 BA FF FF 00 00 3E FF
1050- A9 D3 85 3C A9 03 85 3D
1058- A9 00 85 3E A9 61 85 3F
1060- A9 A0 85 36 A9 1F 85 37
1068- A9 00 85 00 A9 04 85 01
1070- A0 00 A9 00 91 00 E6 00
1078- D0 FA E6 01 A6 01 E0 08
1080- 90 F2 AD 56 C0 AD 52 C0
1088- AD 50 C0 4C FD FE 00 00
1090- 00 00 00 00 00 00 00 00
1098- 00 00 00 00 00 00 00 00
```

表　B — 0

雖然讀入的結果出現 ERR，但這只是程式長度不對而已，並不影響解析的進行。由表 B — 0 來觀察，發現現在（ CSWL、H ）是指向 $0200 ，而磁帶資料起始指位器並未改變，因此 $0200 便是我們目前的 $1200 ，只要將 $1200 以後的程式列出來，便不難明白其內容了。

B—1　程式檢討

將 $1200 以後的內容列出如下：

```
*1200LL

1200-   20 E2 FB    JSR   $FBE2
1203-   A9 00       LDA   #$00
1205-   85 3C       STA   $3C
1207-   85 42       STA   $42
1209-   A9 08       LDA   #$08
120B-   85 3D       STA   $3D
120D-   A9 0B       LDA   #$0B
120F-   85 3F       STA   $3F
1211-   A9 04       LDA   #$04
1213-   85 43       STA   $43
1215-   A9 FF       LDA   #$FF
1217-   85 3E       STA   $3E
1219-   A0 00       LDY   #$00
121B-   AD 56 C0    LDA   $C056
121E-   AD 53 C0    LDA   $C053
1221-   AD 50 C0    LDA   $C050
1224-   20 2C FE    JSR   $FE2C
1227-   A9 00       LDA   #$00
1229-   85 3C       STA   $3C
122B-   A9 08       LDA   #$08
122D-   85 3D       STA   $3D
122F-   A9 FF       LDA   #$FF
1231-   85 3E       STA   $3E
1233-   A9 89       LDA   #$89
1235-   85 3F       STA   $3F
1237-   A9 F8       LDA   #$F8
1239-   85 36       STA   $36
123B-   A9 07       LDA   #$07
123D-   85 37       STA   $37
123F-   A9 00       LDA   #$00
1241-   85 00       STA   $00
1243-   A9 04       LDA   #$04
1245-   85 01       STA   $01
1247-   A0 00       LDY   #$00
1249-   4C FD FE    JMP   $FEFD
124C-   00         BRK
```

程 式　B — 0

這個程式，首先以 JSR $FBE2 使喇叭發出聲音，然後以$FE2C的MOVE副程式來將由（$3C，$3D）及（$3E，$3F）所設定的 $800 至 $BFF 搬動至由（$42，$43）所指定的由 $400 開始的1K螢幕位址內，如此便將某些文字搬動到螢幕了，但這並不是最重要的。在這次的搬動，主要是要將一段重要的進入點位址搬到螢幕位址區！觀察程式B—0發現現在（CSWL、H）是指向 $07F8，所以上述的位址搬動就把程式進入點搬到 $7F8 來了，爲什麼要搬到 $7F8 而不搬到其他位址呢？這是 $7F8～$7FF 是保留給I／O用的特殊位址區，這個區域的內容並不是映像到螢幕上，因此在 $7F8 存放程式才不會在螢幕上出現不必要的圖像。

B—2 保護方式解說

現在將整個過程描述一遍。當第一段保護程式讀入時，同時作了許多動作，包括：①將（CSWL、H）指向 $200 ②將（$3E，$3F）指向 $0BFF，這點相當重要，容後說明③將程式存入 $200 以後的區間④將主程式進入點存入 $BF8～$BFD 區間。

當第一段程式讀入完畢之後，控制權交給COUT之後，便因（CSWL、H）的指向而跳至 $200 去工作了。$200 這段程式如B—1節所述的做完位址搬動的工作，轉換畫面形態，設定主程式的起始、終止位址，將（CSWL、H）指向 $7F8，最後 JMP FEFD的READ副程式將主程式讀入。一旦主程式讀入完畢後，便跳至（CSWL、H）所指向的 $7F8 去了，$7F8 存的指令是:

```
*7F8L

07F8-   20 0C FB    JSR   $FB1E
07FB-   4C 80 1F    JMP   $1F80
07FE-   00          BRK
```

其中 $FB1E 是 PREAD 搖桿按鈕讀取副程式，而 1F80 才是眞正的程式進入點。如果並未依照上述步驟一一解讀，只是將主程讀入，則由於並未將進入點移至 $7F8 ，鍵入 7F8G 程式並不能執行。

B-3 盲目拷貝行不通

如果用前述的盲目拷貝法，直接以 1030.2200R 2200.BFFFR 來讀入程式，然後用 1030.2200W 2200.BFFFW來拷貝，結果並不能成功！

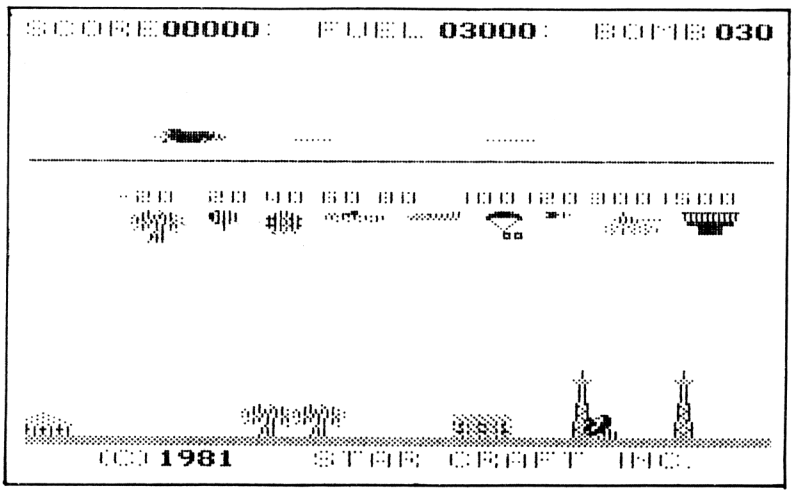
爲什麼與附錄 A "金甲蟲" 結構相似的這個程式，竟然拷貝不成功呢？詳細觀察之後，發現在表 B — 0 當中，第一段保護程式的終止位址是 $0BFF 而不是 $1200 ！那麼我們用 1030.2200R 來讀程式時其實已不正確了，而以 1030.2200W 拷貝程式時，便將 $1C00 到 $2200 這個區間的資料存到卡帶上去了，如此一來當複製出來的卡帶以 30.1200R 讀入程式到 APPLE 時，其實讀到 $0BFF 便已結束，接着自動的便要讀入主程式，但是卡帶上還有多餘的 $0C00 ～ $1200 的資料，如此使得主程式不能讀入到正確位址（前面部份被多餘的這段資料佔住了），當然整個程式是

不能執行了。

那麼如何來解決呢？只要鍵入 1030.1BFFR 1C00.BFFFR 將原來卡帶資料讀進來，然後以 1030.1BFFW 1C00.BFFFW 來複製卡帶，如此便不會造出多餘的資料來了。但是要知道眞正程式的終止位址是 BFF 而不是 1200，除非一一解讀程式，否則是不可能的，所以，盲目拷貝是行不通的！

附錄C 戰略作戰

(200.300R)



C—0 200.300R

現在來看一種用“ 200.300R ”方式保護的程式，這是一種稱爲戰略作戰的卡帶程式。現在先將第一段的保護程式存入 $1200 至 $1300 位址區內，然後解讀其內容：

```
*1200.127F

1200- B2 B0 B0 AE B3 B0 B0 D7
1208- A0 C3 B0 B5 B3 A0 C3 B0
1210- B5 B5 A0 C3 B0 B5 B6 A0
1218- C3 B0 B5 B0 A0 B8 B0 B0
1220- BA C2 C2 A0 B8 B0 B1 BC
1228- B8 B0 B0 AE C2 C6 C6 CD
1230- A0 B8 B0 B0 AE C2 C6 C6
1238- D2 A0 A0 A0 A0 A0 A0 A0
1240- A0 A0 A0 A0 A0 A0 A0 A0
1248- A0 A0 A0 A0 A0 A0 A0 A0
1250- A0 A0 A0 A0 A0 A0 A0 A0
1258- A0 A0 A0 A0 B3 B0 AE B3
1260- C6 C6 D2 8D FE FE 01 01
1268- FE FE 01 01 FE FE 01 01
1270- FE FE 01 01 FE FE 01 01
1278- FE FE 01 01 FE FE 01 01
```

表　C — 0

```
200.300W        C053   C055
C056            C050   800:
BB   801 < 800。BFFM   800.
BFFR  （間隔空白）…… 30.3FFR
```

表　C — 1

由表C — 0將第2頁的ASCII 碼譯出列於表C—1，現在可以發現除了第一段的保護程式之外，還有第二段（800.BFFR）及第三段（30.3FFR），這樣的程式以盲目的拷貝方式當然是不能成功了　。其實第二段的800.BFFR 只是用來保護程式，並無眞正的用途，主程式還是由30.3FFR來讀入的。

C—1　程式檢討

所以找出第三段的前導訊號，以 1030.13FF 來讀取其內容，結果列出如下：

```
*1030.107F

1030- 00 00 FF AA 07 28 40 03
1038- 1B FD 5F 03 3C 03 FF 5F
1040- A9 F0 85 36 A9 FD 85 37
1048- A9 00 85 2A 85 2C A9 40
1050- 85 2B A9 B1 85 2D AD 57
1058- C0 AD 52 C0 AD 50 C0 4C
1060- 9A 03 A0 4B 20 74 03 D0
1068- F9 69 FE B0 F5 A0 21 20
1070- 74 03 C8 C8 88 D0 FD 90
1078- 05 A0 32 88 D0 FD AC 20
```

表　C — 2

由表 C — 2 可以看出 CSWL、H 是指向 $0340 而（$3C，$3D）將接下來的程式轉移錄到 $0340，所以如附錄 A 的金甲蟲的方法，將 $1040 以下的內容列出：

```
*1040LLLLL

1040-   A9 F0      LDA   #$F0
1042-   85 36      STA   $36
1044-   A9 FD      LDA   #$FD
1046-   85 37      STA   $37
1048-   A9 00      LDA   #$00
104A-   85 2A      STA   $2A
104C-   85 2C      STA   $2C
104E-   A9 40      LDA   #$40
1050-   85 2B      STA   $2B
```

```
1052-   A9 B1       LDA   #$B1
1054-   85 2D       STA   $2D
1056-   AD 57 C0    LDA   $C057
1059-   AD 52 C0    LDA   $C052
105C-   AD 50 C0    LDA   $C050
105F-   4C 9A 03    JMP   $039A
1062-   A0 4B       LDY   #$4B
1064-   20 74 03    JSR   $0374
1067-   D0 F9       BNE   $1062
1069-   69 FE       ADC   #$FE
106B-   B0 F5       BCS   $1062
106D-   A0 21       LDY   #$21
106F-   20 74 03    JSR   $0374
1072-   C8          INY
1073-   C8          INY
1074-   88          DEY
1075-   D0 FD       BNE   $1074
1077-   90 05       BCC   $107E
1079-   A0 32       LDY   #$32
107B-   88          DEY
107C-   D0 FD       BNE   $107B
107E-   AC 20 C0    LDY   $C020
1081-   A0 2C       LDY   #$2C
1083-   CA          DEX
1084-   60          RTS
1085-   A5 2A       LDA   $2A
1087-   C5 2C       CMP   $2C
1089-   A5 2B       LDA   $2B
108B-   E5 2D       SBC   $2D
108D-   E6 2A       INC   $2A
108F-   D0 02       BNE   $1093
1091-   E6 2B       INC   $2B
1093-   60          RTS
1094-   20 3A FF    JSR   $FF3A
1097-   4C 00 B0    JMP   $B000
109A-   20 EE 03    JSR   $03EE
109D-   A9 16       LDA   #$16
109F-   20 62 03    JSR   $0362
10A2-   85 0E       STA   $0E
10A4-   20 EE 03    JSR   $03EE
10A7-   A0 24       LDY   #$24
10A9-   20 F1 03    JSR   $03F1
```

```
10AC-   B0 F9       BCS   $10A7
10AE-   20 F1 03    JSR   $03F1
10B1-   A0 3B       LDY   #$3B
10B3-   20 E0 03    JSR   $03E0
10B6-   81 2A       STA   ($2A,X)
10B8-   8D 00 20    STA   $2000
10BB-   45 0E       EOR   $0E
10BD-   85 0E       STA   $0E
10BF-   20 85 03    JSR   $0385
10C2-   A0 35       LDY   #$35
10C4-   90 ED       BCC   $10B3
10C6-   20 E0 03    JSR   $03E0
10C9-   C5 0E       CMP   $0E
10CB-   F0 C7       BEQ   $1094
10CD-   AD 51 C0    LDA   $C051
10D0-   A9 C5       LDA   #$C5
10D2-   20 ED FD    JSR   $FDED
10D5-   A9 D2       LDA   #$D2
10D7-   20 ED FD    JSR   $FDED
10DA-   20 ED FD    JSR   $FDED
10DD-   4C 94 03    JMP   $0394
10E0-   A2 08       LDX   #$08
10E2-   48          PHA
10E3-   20 EE 03    JSR   $03EE
10E6-   68          PLA
10E7-   2A          ROL
10E8-   A0 3A       LDY   #$3A
10EA-   CA          DEX
10EB-   D0 F5       BNE   $10E2
10ED-   60          RTS
10EE-   20 F1 03    JSR   $03F1
10F1-   88          DEY
10F2-   AD 60 C0    LDA   $C060
10F5-   45 0F       EOR   $0F
10F7-   10 F8       BPL   $10F1
10F9-   45 0F       EOR   $0F
10FB-   85 0F       STA   $0F
10FD-   C0 80       CPY   #$80
10FF-   60          RTS
```

程式　C—0

程式C — 0 主要是將 APPLE監督程式裡的READ副程式 加以重寫而已，仔細與參考手册的監督程式對照，可以將相關的副程式依位址順序列出如下：

副　程　式	
HEADR	$0362
WRTAPE	$0377
NXTA1	$0385
BELL	$FF3A
READ	$039A
RDBYTE	$03E0
RDBYT2	$03E2
RD2BIT	$03EF
RDBIT	$03F1

表　C — 3

零　頁　位　址	
CHKSUM	$0E
LASTIN	$0F
A1L	$2A
A1H	$2B
A2L	$2C
A2H	$2D

表　C — 4

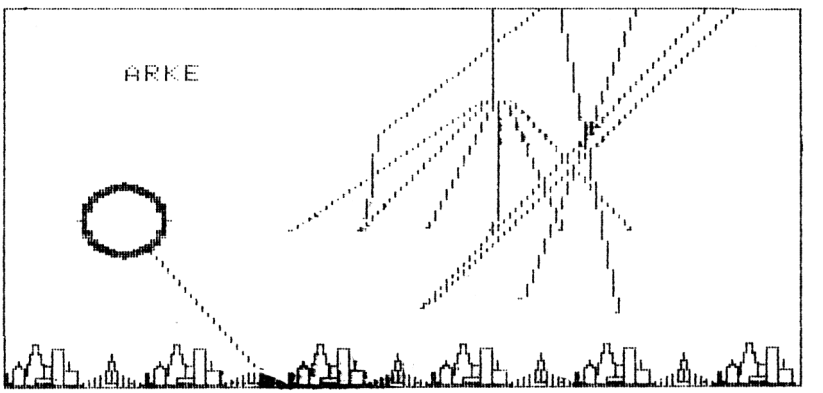
因此程式C — 0 幾乎完全與READ副程式相同，唯一差別的是$03B8（即$10B8）的STA $2000，這個指令是接在位元組讀取指令STA（$2A，X）之後，因此每次由磁帶讀入一個位元組同時也將資料存到$2000，而$2000是高解像畫面最左上角的位址，因此，在讀取過程當中，可以看到左上角7個不斷變化的閃動亮點，這表示程式正在讀取。另外，幾個重要的位址是（$2A，$2B）及（$2C，$2D）這四個位址分別取代原有的（$3C，$3D）及（$3E，$3F），因此程式的起始位址爲由$348～$0355所設定的$4000至$B100，而程式進入點則爲$0397的$B000。

C—2 保護方式解說

這個程式總共有4段，亦即①最初的200.300R②表C — 1的800.BFFR③30.3FFR及④由$0348至$3FF的READ所讀入的4000.B100R。整個過程並不複雜，只是共有4段的錄音，如果不一步步分析，隨便拷貝也是不易成功的，唯一的一點缺憾是重寫的READ副程式原樣照抄，如果能稍加改變，則更具保護效果。至於如何改變READ副程式，則可依照第4章的原則來加以修改，下一個例子便是很簡單的將HEADR修改，而使保護更加周全。

附錄D 攻 陷 地 球

(200.300R)



D—0 200.300R

依照前述原理，將第一段程式錄到 $200 至 $1300 區間，然後解讀其內容：

```
*1200.124F

1200- B2 B0 B0 AE B1 B2 B0 B0
1208- A0 A0 C3 B0 B5 B2 A0 C3
1210- B0 B5 B4 A0 C3 B0 B5 B7
1218- A0 C3 B0 B5 B0 A0 A0 A0
1220- B3 B0 B0 AE B6 B2 C6 C6
1228- D2 A0 B6 B2 B0 B0 C7 8D
1230- A0 A0 A0 A0 A0 A0 A0 A0
1238- A0 A0 A0 A0 A0 A0 A0 A0
1240- A0 A0 A0 A0 A0 A0 A0 A0
1248- A0 A0 A0 A0 A0 A0 A0 A0
```

表　D — 0

200.1200	C052	C054
C057	C050	300.
62FFR	6200G	RETURN

表　D — 1

結果發現，內容很簡單，只要將接下來的主程式錄到 $300 至 $62FF 便可以了。當主程式錄完之後，便鍵入 6200G 來執 行程式，結果？螢幕竟然沒有任何動作！

由於至目前爲止，我們的方法並無錯誤，所以可以推論的是，問題一定是出在 6200G 以後，將 $6200 以後的內容列出如下：

```
*6200LL

6200-   A2 36       LDX   #$36
6202-   86 3C       STX   $3C
6204-   A2 37       LDX   #$37
6206-   86 3E       STX   $3E
6208-   A2 00       LDX   #$00
620A-   86 3D       STX   $3D
620C-   86 3F       STX   $3F
620E-   20 FA FC    JSR   $FCFA
6211-   A9 02       LDA   #$02
6213-   20 C9 FC    JSR   $FCC9
6216-   85 2E       STA   $2E
6218-   20 FA FC    JSR   $FCFA
621B-   A0 24       LDY   #$24
621D-   20 FD FC    JSR   $FCFD
6220-   B0 F9       BCS   $621B
6222-   20 FD FC    JSR   $FCFD
6225-   A0 3B       LDY   #$3B
6227-   20 EC FC    JSR   $FCEC
622A-   81 3C       STA   ($3C,X)
622C-   45 2E       EOR   $2E
622E-   85 2E       STA   $2E
6230-   20 BA FC    JSR   $FCBA
6233-   A0 35       LDY   #$35
6235-   90 F0       BCC   $6227
6237-   20 EC FC    JSR   $FCEC
623A-   C5 2E       CMP   $2E
623C-   6C 36 00    JMP   ($0036)
```

程 式　D — 0

原來這個程式是將（$3C，$3D）、（$3E，$3F）磁帶起始、終止指位器設定成$36及$37，也就是由6200G以後要做的，是用來將兩個位元組的資料讀入到（$36，$37），這樣做有什麼用意呢？原來（$36，$37）＝（CSWL、H）是字元輸出指位器，一旦READ副程式執行完畢，便會往（CSWL、H）所指的方

向去執行任務，所以可以得知的，現在（$36，$37）所讀入的應該是眞正的程式進入點，而$6200只不過是用來讀入進入點的一段程式而已。

依此推論，$6200以下的程式必然是磁帶讀取副程式了，除了上述的起始，終止位址設定外，其中一些副程式分別是：

副　程　式	
RD2BIT	$FCFA
HEADR	$FCC9
RDBIT	$FCFD
RDBYTE	$FCEC
NXTA1	$FCBA

表　D－2

將表D—2與程式D—0對照，發現基本上程式D—0與READ副程式並無不同，將參考手册169頁（英文）的READ副程式與程式D—0比較，除了最後的JMP（$0036）之外，只有$6211的LDA #$02差別而已。由第4章知道，JSR HEADR之前必須設定A—暫存器的值，A—暫存器的值決定HEADR的長度，現在HEADR的長度＝2×256×1毫秒＝0.5秒，傾聽第三段（緊接主程式之後）程式的前導訊號也是相當的短，這是WRITE副程式也經過修改之故。對於第三段程式如果以一般的READ副程式來讀取是不能成功的，我們可以試着以0.1R來讀取第三段的兩位元組資料，結果由於HEADR相差太多而不成功。

D—1　保護方式解說

現在將程式保護方法描述一遍。首先利用第2頁區來存放主程式起始、終止位址，並使之自動讀入。當主程式讀入完畢之後，便自動的以6200G來將程式進入點位址讀至（$36，$37）指位器上，最後以JMP（$0036）來執行程式。

如果找到第三段程式，以0.1R來讀取進入位址是不能成功的，這是由於上述的HEADR長度不一致的緣故，當然以盲目拷貝的方式，用1200.1300R　1301.BFFFR來讀入程式的話，便會因爲遺漏第三段的進入位址而不能動作。唯一的方法，是利用程式本身來讀取進入位址，也就是說先將主程式讀入，然後以6200G來讀取進入位址，便可以了。但是如此的話，會因爲JMP（$0036）而使程式自動執行，要解決這個問題，只要將$6201及$6205的（$36，$37）改成（$00，$01）便可解決，現在鍵入：

```
*6201:00
*6205:01
```

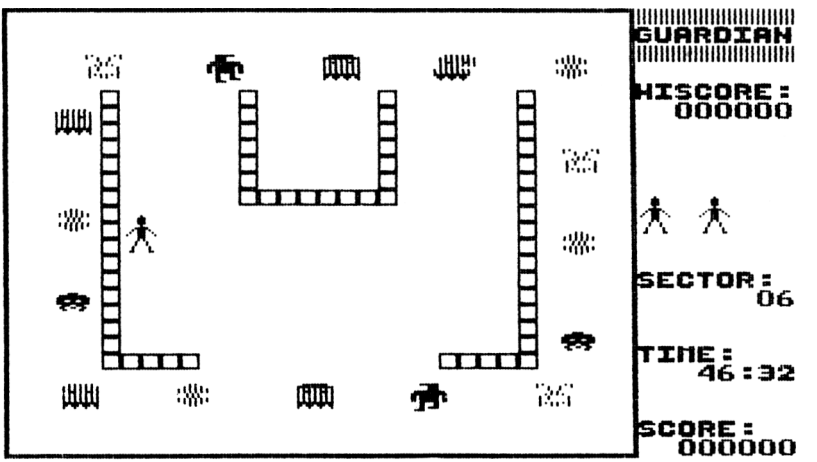
接着拔掉錄音綫聽到第三段程式的前導訊號（注意很短！）時立即按STOP，然後鍵入6200G（不要RETURN），當一按下錄音機PLAY鍵之後，立即按下RETURN鍵，便可成功的將進入位址讀到（$00，$01）裡頭了，現在鍵入：

```
*0.1
0000- E3 58
```

我們可以看出，原來進入位址在 $58E3 ，現在鍵入 58E3G 便可以執行程式了，以本例子的程式來說，按下RESET後，再執行也可順利的動作。這種保護方式，顯然比附錄C的方式更好，它利用HEADR　來改變READ副程式的動作，如此使得正常的讀取方式無法正確的取得進入點位址。

附錄E 世 界 末 日

(200.2FFR)



E—0 200.2FFR

接下來要介紹的保護方式，較不同於前述幾個例子，首先將第一段程式讀入到 $1200 至 $12FF 的區間，並列出解讀：

```
*1200.127F

1200- 1D 43 AE B4 B0 D2 A0 A0
1208- A0 A0 A0 A0 A0 B2 C3 B0
1210- C7 A0 B4 B0 B0 AE B5 B0
1218- B0 B0 D2 A0 B3 B4 B0 B0
1220- AE B5 B6 B0 B0 D2 A0 B2
1228- C3 B0 C7 A0 B2 B0 B0 AE
1230- B2 C6 C6 D2 A0 B2 C3 B0
1238- C7 A0 B4 B0 B0 AE B5 B0
1240- B0 B0 D2 A0 B6 B0 B0 B0
1248- BC B2 B8 B0 B0 AE B5 B6
1250- B0 B0 CD A0 B4 B0 B0 B0
1258- BC B8 B0 B0 AE B2 B8 B0
1260- B0 CD A0 A0 B4 B0 B0 B0
1268- C7 A0 B6 A0 C3 B0 B5 B0
1270- 20 00 BA A9 00 85 4A A9
1278- 08 85 4B A9 00 85 4C A9
```

表　E — 0

由表E — 0發現首先要執行一個2C0G的動作，現在將$2C0以後的程式內容列出：

```
12C0-   AD 00 02    LDA   $0200
12C3-   85 34       STA   $34
12C5-   AD 53 C0    LDA   $C053
12C8-   AD 56 C0    LDA   $C056
12CB-   AD 50 C0    LDA   $C050
12CE-   20 58 FC    JSR   $FC58
12D1-   20 36 F8    JSR   $F836
12D4-   A9 01       LDA   #$01
12D6-   8D C1 02    STA   $02C1
12D9-   A9 60       LDA   #$60
12DB-   8D C5 02    STA   $02C5
12DE-   60          RTS
```

程式　E — 0

這個程式看似沒什麼多大作用，它首先將 $200 的內含值取入 $34，然後觸動畫面開關，清除畫面，最後將 #$01 及#$60 存入 $2C1 及 $2C5，所以執行後 $2C0 的程式變成：

```
*12C0L

12C0-   AD 01 02    LDA   $0201
12C3-   85 34       STA   $34
12C5-   60          RTS
12C6-   53          ???
```

程式　E — 1

再對照表 E — 0，接下來動作是：400.5000R　3400.5600R　2C0G　200.2FFR……等一大串的動作，依照這些動作一一執行結果……？

E-1 程式檢討

爲什麼這個程式的解讀，看似簡單，結果却不能執行呢？而 $2C0 這個程式有何作用呢？

$2C0 這個程式唯一令人不瞭解其用意的，便是 STA　$34 這個動作，$34 是什麼位址呢？關鍵必然在此！查看 APPLE 監督程式的內容，發現其中有一個指令讀取副程式群，其中一個副程式稱爲 NXTCHR = $FFAD，這個副程式是：

```
*FFADL

FFAD-   B9 00 02    LDA   $0200,Y
FFB0-   C8          INY
FFB1-   49 B0       EOR   #$B0
FFB3-   C9 0A       CMP   #$0A
FFB5-   90 D3       BCC   $FF8A
FFB7-   69 88       ADC   #$88
FFB9-   C9 FA       CMP   #$FA
FFBB-   B0 CD       BCS   $FF8A
FFBD-   60          RTS
```

程 式　E — 2

在 $FFAD副程式中，第一句話是LDA　$200，Y，也就是說以Y暫存器的內含值做索引來讀取第 2 頁鍵盤緩衝區的內容，所以只要改變Y——暫存器的內含值，便可以使NXTCHR 讀取不同的緩衝區內的資料。可是如何改變Y——暫存器的值呢？再看APPLE監督程式中另一個與NXTCHR的相關副程式CHRSRCH = $FF7A：

```
*FF7AL

FF7A-   88          DEY
FF7B-   30 E8       BMI   $FF65
FF7D-   D9 CC FF    CMP   $FFCC,Y
FF80-   D0 F8       BNE   $FF7A
FF82-   20 BE FF    JSR   $FFBE
FF85-   A4 34       LDY   $34
FF87-   4C 73 FF    JMP   $FF73
```

程 式　E — 3

其中 $FF85　是　LDY　$34　，原來 Y——暫存器內含值是由 $34 來設定的！至此，相信一切已明白了，在程式E — 0 及E — 1 中便是設定 $34（YSAV）這個零頁位址，如此使得 CHRSRCH 及 NXTCHR 在讀取程式時？可以不依順序而是由 $2C0 這個程式來設定其在第 2 頁區讀取位置，如果 $34 的值設定成 $00，NXTCHR 便讀取 $0200，如果 $34 的值設定爲 $FF，NXTCHR 便讀取 $02FF 的資料。

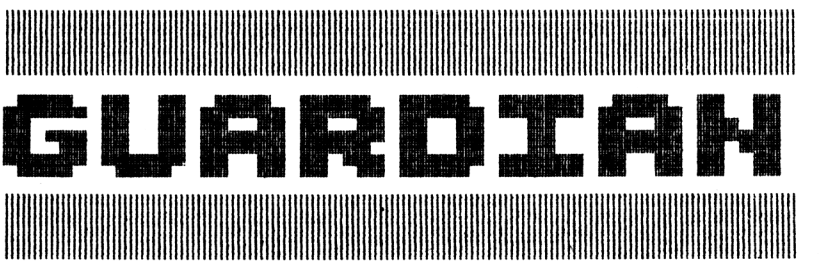
E-2 保護方式解說

現在一切終於明白，首先依表 E — 0 的順序，先執行 2C0G，結果將 $200 的內含值存入 $34，也就是說 Y——暫存器是被設定成 #$1D，依照程式 E — 2 可以看出 APPLE 接下來要做的不是 2C0G 後面的 400.5000R 而是由程式 E — 2 的 LDA　$0200，Y 所指定位址爲 $21D 的 400.5600R，這便是眞正主程式的位址，如果不依此解讀必然會以爲是 3400.5600R 而使得解讀失敗了。

主程式讀完之後，又一次 2C0G（查表 E — 0），這次的 2C0G 執行程式 E — 1 的動作，結果將 $201 的內含值 #$43 取入 $34，也就是將 Y——暫存器設定成 #$43，因此 NXTCHR 接下來是讀取 $0243 的內容，$243 以下的內容是 6000 < 2800.5600M　4000 < 800.2800M　4000G，依次將上列動作執行，最後的 $4000，便是眞正的程式進入點了。

至此，連同本書第 0 章及第一册的 200.1200R 總共介紹 7 種程式保護法，我們發現，一切唯由從基本着手，追踪 APPLE 內

部動作，才能完全掌握整體運作情形及原理，盲目的拷貝與道聽途說是不可靠的。如果能夠徹底瞭解APPLE的基本原理，則未嘗不能自己創造出更具功能的系統吧！



* 有關APPLE系統解析，請參閱第三册。

附 錄 F

3 種 BASIC 保護法

有關BASIC 程式的保護方式，也是如機械語言方式一般的種類繁多，欲徹底瞭解其運作情形，必須如前述幾種保護方式一般的詳予追踪研析才行，可是這又牽涉到APPLE的BASIC解譯器，因此留至後書詳述，此處只列出幾種方式，使讀者能用來保護自己的程式。

BASIC Ⅰ：POKE 2049，0

```
]LIST

0  REM   *** THIS IS A DEMON PROG
      RAM ***
10  HOME
20  VTAB 10: HTAB 15: PRINT "***
      ABC ***"
30  REM   *** PLEASE KEY IN 'POKE
       2049,0' ***

] POKE 2049,0
```

程 式 F — 0

請鍵入程式F — 0 ，然後POKE 2049，0 ，結果程式可以執行，但列出時，只有第 0 行：

```
]RUN
                *** ABC ***

]LIST

0  REM   *** THIS IS A DEMON PROG
     RAM ***
```

BASIC Ⅱ： POKE 214，128

```
]LIST

0  REM  *** THIS IS A DEMON PROGR
     AM ***
5  GR
10  FOR I = 0 TO 15
20  COLOR= I
30  VLIN 0,39 AT I * 2
40  NEXT I
50  REM   *** PLEASE KEY IN 'POKE
      214,128' ***
```

程 式 F — 1

鍵入上列程式，先執行一遍，然後鍵入POKE 214，128，接下來只要鍵入任何指令，均變成RUN而使程式自動執行，當然也LIST不出來了。

BASIC Ⅲ： POKE 82，213

```
]LIST

10  REM  *** THIS IS A DEMON PROG
     RAM ***
20  HGR
30  FOR I = 1 TO 7
40  HCOLOR= I
50  HPLOT 0,0
60  CALL 62454
70  FOR T = 0 TO 1000: NEXT T
80  NEXT I
90  REM  *** PLEASE KEY IN 'POKE
     82,213' AND THEN SAVE THE PR
     OGRAM. WHEN YOU LOAD THE PRO
     GRAM IT WILL RUN AUTOMATICAL
     LY ***
```

程 式　F — 2

鍵入程式F — 2，然後POKE　82，213，接着將程式SAVE到磁帶上，當以LOAD讀回程式時，便會自動執行了，而且RESET後，無論鍵入任何指令，均當成RUN而自動執行。

附錄G

6502指令表

THE FOLLOWING NOTATION APPLIES TO THIS SUMMARY:

A	Accumulator
X, Y	Index Registers
M	Memory
C	Borrow
P	Processor Status Register
S	Stack Pointer
✓	Change
—	No Change
+	Add
∧	Logical AND
-	Subtract
∀	Logical Exclusive Or
↑	Transfer From Stack
↓	Transfer To Stack
←	Transfer To
→	Transfer To
∨	Logical OR
PC	Program Counter
PCH	Program Counter High
PCL	Program Counter Low
OPER	Operand
#	Immediate Addressing Mode

FIGURE 1. ASL-SHIFT LEFT ONE BIT OPERATION

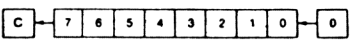


FIGURE 2. ROTATE ONE BIT LEFT (MEMORY OR ACCUMULATOR)

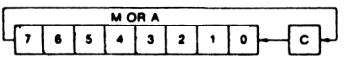
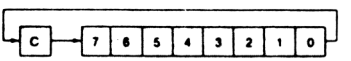


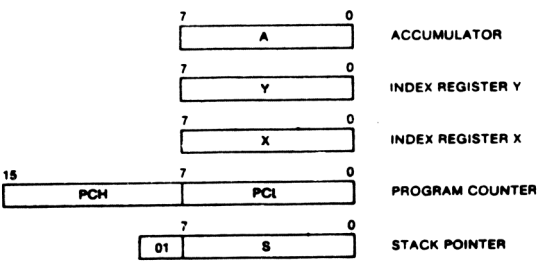
FIGURE 3

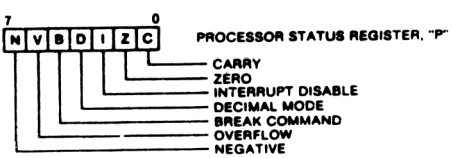


NOTE 1: BIT — TEST BITS

Bit 6 and 7 are transferred to the status register. If the result of A ∧ M is zero then Z=1, otherwise Z=0

PROGRAMMING MODEL





INSTRUCTION CODES

Name Description	Operation	Addressing Mode	Assembly Language Form	HEX OP Code	No. Bytes	"P" Status Reg N Z C I D V
ADC						
Add memory to accumulator with carry	A+M+C → A C	Immediate	ADC #Oper	69	2	√√√- -√
		Zero Page	ADC Oper	65	2	
		Zero Page,X	ADC Oper,X	75	2	
		Absolute	ADC Oper	6D	3	
		Absolute,X	ADC Oper,X	7D	3	
		Absolute,Y	ADC Oper,Y	79	3	
		(Indirect,X)	ADC (Oper,X)	61	2	
		(Indirect),Y	ADC (Oper),Y	71	2	
AND						
"AND" memory with accumulator	A A M → A	Immediate	AND #Oper	29	2	√√- - - -
		Zero Page	AND Oper	25	2	
		Zero Page,X	AND Oper,X	35	2	
		Absolute	AND Oper	2D	3	
		Absolute,X	AND Oper,X	3D	3	
		Absolute,Y	AND Oper,Y	39	3	
		(Indirect,X)	AND (Oper,X)	21	2	
		(Indirect),Y	AND (Oper),Y	31	2	
ASL						
Shift left one bit (Memory or Accumulator)	(See Figure 1)	Accumulator	ASL A	0A	1	√√√ - -
		Zero Page	ASL Oper	06	2	
		Zero Page,X	ASL Oper,X	16	2	
		Absolute	ASL Oper	0E	3	
		Absolute,X	ASL Oper,X	1E	3	
BCC						
Branch on carry clear	Branch on C=0	Relative	BCC Oper	90	2	- - - - - -
BCS						
Branch on carry set	Branch on C=1	Relative	BCS Oper	B0	2	- - - - - -
BEQ						
Branch on result zero	Branch on Z=1	Relative	BEQ Oper	F0	2	- - - - - -
BIT						
Test bits in memory with accumulator	A A M M_7 → N, M_6 → V	Zero Page	BIT* Oper	24	2	M_7√- - - M_6
		Absolute	BIT* Oper	2C	3	
BMI						
Branch on result minus	Branch on N=1	Relative	BMI Oper	30	2	- - - - - -
BNE						
Branch on result not zero	Branch on Z=0	Relative	BNE Oper	D0	2	- - - - - -
BPL						
Branch on result plus	Branch on N=0	Relative	BPL oper	10	2	- - - - - -
BRK						
Force Break	Forced Interrupt PC+2 ↓ P ↓	Implied	BRK*	00	1	- - - 1 - -
BVC						
Branch on overflow clear	Branch on V=0	Relative	BVC Oper	50	2	- - - - - -

Name Description	Operation	Addressing Mode	Assembly Language Form	HEX OP Code	No. Bytes	"P" Status Reg. N Z C I D V
BVS Branch on overflow set	Branch on V=1	Relative	BVS Oper	70	2	------
CLC Clear carry flag	0 → C	Implied	CLC	18	1	---0--
CLD Clear decimal mode	0 → D	Implied	CLD	D8	1	-0----
CLI	0 → I	Implied	CLI	58	1	---0--
CLV Clear overflow flag	0 → V	Implied	CLV	B8	1	0-----

CMP
Compare memory and accumulator
	A — M	Immediate
		Zero Page
		Zero Page, X
		Absolute
		Absolute,X
		Absolute,Y
		(Indirect,X)
		(Indirect),Y
			CMP #Oper
			CMP Oper
			CMP Oper,X
			CMP Oper
			CMP Oper,X
			CMP Oper,Y
			CMP (Oper,X)
			CMP (Oper),Y
				C9
				C5
				D5
				CD
				DD
				D9
				C1
				D1
					2
					2
					2
					3
					3
					3
					2
					2
						√√√---

CPX Compare memory and index X	X — M	Immediate Zero Page Absolute	CPX #Oper CPX Oper CPX Oper	E0 E4 EC	2 2 3	√√√---
CPY Compare memory and index Y	Y — M	Immediate Zero Page Absolute	CPY #Oper CPY Oper CPY Oper	C0 C4 CC	2 2 3	√√√---
DEC Decrement memory by one	M — 1 → M	Zero Page Zero Page,X Absolute Absolute,X	DEC Oper DEC Oper,X DEC Oper DEC Oper,X	C6 D6 CE DE	2 2 3 3	√√----
DEX Decrement index X by one	X — 1 → X	Implied	DEX	CA	1	√√----
DEY Decrement index Y by one	Y — 1 → Y	Implied	DEY	88	1	√√----

Name Description	Operation	Addressing Mode	Assembly Language Form	HEX OP Code	No. Bytes	"P" Status Reg. N Z C I B V
EOR						
"Exclusive-Or" memory with accumulator	A V M → A	Immediate	EOR #Oper	49	2	√√ ---
		Zero Page	EOR Oper	45	2	
		Zero Page,X	EOR Oper,X	55	2	
		Absolute	EOR Oper	4D	3	
		Absolute,X	EOR Oper,X	5D	3	
		Absolute,Y	EOR Oper,Y	59	3	
		(Indirect,X)	EOR (Oper,X)	41	2	
		(Indirect),Y	EOR (Oper),Y	51	2	
INC						
Increment memory by one	M + 1 → M	Zero Page	INC Oper	E6	2	√√----
		Zero Page,X	INC Oper,X	F6	2	
		Absolute	INC Oper	EE	3	
		Absolute,X	INC Oper,X	FE	3	
INX						
Increment index X by one	X + 1 → X	Implied	INX	E8	1	√√----
INY						
Increment index Y by one	Y + 1 → Y	Implied	INY	C8	1	√√----
JMP						
Jump to new location	(PC+1) → PCL (PC+2) → PCH	Absolute Indirect	JMP Oper JMP (Oper)	4C 6C	3 3	------
JSR						
Jump to new location, saving return address	PC+2 ↓ (PC+1) → PCL (PC+2) → PCH	Absolute	JSR Oper	20	3	------
LDA						
Load accumulator with memory	M → A	Immediate	LDA #Oper	A9	2	√√----
		Zero Page	LDA Oper	A5	2	
		Zero Page,X	LDA Oper,X	B5	2	
		Absolute	LDA Oper	AD	3	
		Absolute,X	LDA Oper,X	BD	3	
		Absolute,Y	LDA Oper,Y	B9	3	
		(Indirect,X)	LDA (Oper,X)	A1	2	
		(Indirect),Y	LDA (Oper),Y	B1	2	
LDX						
Load index X with memory	M → X	Immediate	LDX #Oper	A2	2	√√----
		Zero Page	LDX Oper	A6	2	
		Zero Page,Y	LDX Oper,Y	B6	2	
		Absolute	LDX Oper	AE	3	
		Absolute,Y	LDX Oper,Y	BE	3	
LDY						
Load index Y with memory	M → Y	Immediate	LDY #Oper	A0	2	√√----
		Zero Page	LDY Oper	A4	2	
		Zero Page,X	LDY Oper,X	B4	2	
		Absolute	LDY Oper	AC	3	
		Absolute,X	LDY Oper,X	BC	3	

Name Description	Operation	Addressing Mode	Assembly Language Form	HEX OP Code	No. Bytes	"P" Status Reg. N Z C I D V
LSR Shift right one bit (memory or accumulator)	(See Figure 1)	Accumulator Zero Page Zero Page,X Absolute Absolute,X	LSR A LSR Oper LSR Oper,X LSR Oper LSR Oper,X	4A 46 56 4E 5E	1 2 2 3 3	0√√----
NOP No operation	No Operation	Implied	NOP	EA	1	------
ORA "OR" memory with accumulator	A V M → A	Immediate Zero Page Zero Page,X Absolute Absolute,X Absolute,Y (Indirect,X) (Indirect),Y	ORA #Oper ORA Oper ORA Oper,X ORA Oper ORA Oper,X ORA Oper,Y ORA (Oper,X) ORA (Oper),Y	09 05 15 0D 1D 19 01 11	2 2 2 3 3 3 2 2	√√----
PHA Push accumulator on stack	A ↓	Implied	PHA	48	1	------
PHP Push processor status on stack	P ↓	Implied	PHP	08	1	------
PLA Pull accumulator from stack	A ↑	Implied	PLA	68	1	√√----
PLP Pull processor status from stack	P ↑	Implied	PLP	28	1	From Stack
ROL Rotate one bit left (memory or accumulator)	(See Figure 2)	Accumulator Zero Page Zero Page,X Absolute Absolute,X	ROL A ROL Oper ROL Oper,X ROL Oper ROL Oper,X	2A 26 36 2E 3E	1 2 2 3 3	√√√---
ROR Rotate one bit right (memory or accumulator)	(See Figure 3)	Accumulator Zero Page Zero Page,X Absolute Absolute,X	ROR A ROR Oper ROR Oper,X ROR Oper ROR Oper,X	6A 66 76 6E 7E	1 2 2 3 3	√√√---

Name Description	Operation	Addressing Mode	Assembly Language Form	HEX OP Code	No. Bytes	"P" Status Reg N Z C I D V
RTI Return from interrupt	P↓PC↓	Implied	RTI	40	1	From Stack
RTS Return from subroutine	PC↓, PC+1 → PC	Implied	RTS	60	1	------
SBC Subtract memory from accumulator with borrow	A - M - $\overline{C}$ → A	Immediate	SBC #Oper	E9	2	√√√--√
		Zero Page	SBC Oper	E5	2	
		Zero Page,X	SBC Oper,X	F5	2	
		Absolute	SBC Oper	ED	3	
		Absolute,X	SBC Oper,X	FD	3	
		Absolute,Y	SBC Oper,Y	F9	3	
		(Indirect,X)	SBC (Oper,X)	E1	2	
		(Indirect),Y	SBC (Oper),Y	F1	2	
SEC Set carry flag	1 → C	Implied	SEC	38	1	--1---
SED Set decimal mode	1 → D	Implied	SED	F8	1	----1-
SEI Set interrupt disable status	1 → I	Implied	SEI	78	1	---1--
STA Store accumulator in memory	A → M	Zero Page	STA Oper	85	2	------
		Zero Page,X	STA Oper,X	95	2	
		Absolute	STA Oper	8D	3	
		Absolute,X	STA Oper,X	9D	3	
		Absolute,Y	STA Oper,Y	99	3	
		(Indirect,X)	STA (Oper,X)	81	2	
		(Indirect),Y	STA (Oper),Y	91	2	
STX Store index X in memory	X → M	Zero Page	STX Oper	86	2	------
		Zero Page,Y	STX Oper,Y	96	2	
		Absolute	STX Oper	8E	3	
STY Store index Y in memory	Y → M	Zero Page	STY Oper	84	2	------
		Zero Page,X	STY Oper,X	94	2	
		Absolute	STY Oper	8C	3	
TAX Transfer accumulator to index X	A → X	Implied	TAX	AA	1	√√----
TAY Transfer accumulator to index Y.	A → Y	Implied	TAY	A8	1	√√----
TSX Transfer stack pointer to index X	S → X	Implied	TSX	BA	1	√√----

Name Description	Operation	Addressing Mode	Assembly Language Form	HEX OP Code	No. Bytes	"P" Status Reg. N Z C I D V
TXA Transfer index X to accumulator	X → A	Implied	TXA	8A	1	√√----
TXS Transfer index X to stack pointer	X → S	Implied	TXS	9A	1	------
TYA Transfer index Y to accumulator	Y → A	Implied	TYA	98	1	√√----

參 考 書 目

1. SYSTEM Description : The Apple Ⅱ S. Wozniak, BYTE, May, 1977.
2. The Apple Ⅱ Monitor Peeled, William E. Dougherty
3. Apple Ⅱ Reference Manual (Apple Computer Inc.)
4. 6500 Hardware & Programming Manual (Rockwell Inc.)
5. CALL APPLE (One of the Apple user group newsletters)
6. Nibble express (One of the Apple user group newsletters)
7. PA——EDU80 手册 (全亞)
8. RCA COSMAC VIP—311 (RCA Corporation)
9. Microprocessor Application Handbook David F. Stout (Mcgraw-Hill)
10. The PET Revealed (Computabits Ltd.)

H. K. $ 25.00

Published & Printed
in Hong Kong

APPLE II 微電腦徹底研究(二) 進階篇 施威銘著